Naval Research Laboratory

Washington, DC 20375-5320

NRL/FR/5591--12-10,224

Exploiting the Multi-Service Domain Protecting Interface

Christopher L. Robson

Center for Computational Science
Information Technology Division

October 17, 2012

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*
17-10-2012

2. REPORT TYPE
Formal Report

3. DATES COVERED *(From - To)*
1 October 2010 – 30 September 2011

4. TITLE AND SUBTITLE
Exploiting the Multi-Service Domain Protecting Interface

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
0603011F

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
J955

6. AUTHOR(S)
Christopher L. Robson

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Research Laboratory
4555 Overlook Avenue, SW
Washington, DC 20375-5320

8. PERFORMING ORGANIZATION REPORT NUMBER
NRL/FR/5591--12-10,224

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
SAF/FMBIB-AFOY
P.O. Box 14200
Washington, DC 20044-4200

10. SPONSOR / MONITOR'S ACRONYM(S)
SAF/FMBIB-AFOY

11. SPONSOR / MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The technology discussed in this paper demonstrates how a single converged control plane can benefit the U.S. government. It converges the capabilities of information assurance and policy with inter- and intra-domain routing and protection into a single control plane. Additionally, this technology provides a single utility which integrates communications and management control functions. It can asynchronously execute applications locally and/or remotely. This design is attractive to the U.S. government because of its ability to converge and centrally control various functions into one common control plane.

15. SUBJECT TERMS
Session Initiation Protocol
Control plane
Network encryption device
Routing

16. SECURITY CLASSIFICATION OF: a. REPORT	b. ABSTRACT	c. THIS PAGE	17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
Unclassified	Unclassified	Unclassified	Unlimited	89	Christopher L. Robson
					19b. TELEPHONE NUMBER *(include area code)* 202-404-3138

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

This page
intentionally
left blank

CONTENTS

FIGURES

TABLES

ACRONYMS AND ABBREVIATIONS

API	Application Programming Interface
ATDNet	Advanced Technology Demonstration Network
ATM	Asynchronous Transfer Mode
CT	Cypher Text
CTD	Cypher Text Domain
DDR	Double Data Rate
DISN	Defense Information Systems Network
DoD	Department of Defense
EE	Encryption Engine
FEON	Fast Encrypting Operational Networks
FIFO	First In First Out
FPGA	Field Programmable Gate Array
GE	Gigabit Ethernet
GRH	Global Routing Header
HAIPE	High Assurance Internet Protocol Encryptor
HSET	High Speed Encryption Technology
IA	Information Assurance
IB	InfiniBand
IC	Intelligence Community
I/O	Input/Output
IP/TCP	Internet Protocol/T
I-R	Initiator-Responder
ISSO	Information Systems Security Officer
ISP	Internet Service Provider
JTCD	Joint Technology Capability Demonstration
L2VPN	Layer 2 Virtual Private Network
LDC	Local Domain Control
LER	Label Edge Router
LRH	Local Routing Header
LSP	Label Switched Path
LSR	Label Switch Router
MIPR	Military Interdepartmental Purchase Request
MPLS	MultiProtocol Label Switch
MSDPI	Multi-Service Domain Protecting Interface
NMS	Network Management System
NRL	Naval Research Laboratory
NTP	Network Time Protocol
OFED	Open Fabrics Enterprise Distribution
OS	Operating System
P2P	Peer to Peer
PCIe	Peripheral Component Interconnect Express
PDC	Public Domain Control

PEP	Presence Event Package
PIDF	Presence Information Data Format
POC	Point of Contact
PT	Plain Text
PTD	Plain Text Domain
PWE	Pseudo Wire Emulation
QDR	Quad Data Rate
QoS	Quality of Service
QP	Queue Pair
R&D	Research and Development
RED	Random Early Detection
RIB	Routing Information Base
SA	Security Association
SIMPLE	SIP for Instant Messaging and Presence Leveraging Extensions
SIP	Session Initiation Protocol
SSL	Secure Sockets Layer
T&E	Test and Evaluation
TLS	Transport Layer Security
URI	Uniform Resource Identifier
VLAN	Virtual Local Area Network
VoIP	Voice Over Internet Protocol
VPN	Virtual Private Network
VRF	Virtual Routing and Forwarding
VRFB	Virtual Routing and Forwarding Buffer
WAN	Wide Area Network
XML	Extensible Markup Language

EXPLOITING THE MULTI-SERVICE DOMAIN PROTECTING INTERFACE

1 INTRODUCTION

This report presents details of the work performed by the Naval Research Laboratory (NRL) to meet tasking obligations designated in MIPR M448514, Fast Encrypting Operational Networks High Speed Encryption Technology (FEON HSET). The goal of the FEON HSET program was to develop a high speed network encryptor. NRL was tasked to provide support to sponsor-designated partner laboratories in the areas of networks, systems, and applications research, and engineering and testing. This report focuses on the following work conducted.

1. InfiniBand (IB) testing was performed between two systems. InfiniBand tools and systems were modified as required to successfully complete the IB testing. A high speed network infrastructure, the FEON HSET test bed, was put in place to support all testing between participants. Where this network infrastructure was not possible, test data was collected using the resources available at NRL.

2. Experiments were conducted that show how to exploit the Multi-Service Domain Protecting Interface (MSDPI) architecture [1] to meet the tasking set in the MIPR. MSDPI is a new communications interface used by encryption devices and network devices as a routing engine or test tool suite allowing Department of Defense (DoD) and intelligence community (IC) networks to keep up with today's rapid advances in technology and continuous changes in threats without requiring modification or recertification. For example, the experiments discussed in this report show how an encryption device exploiting the MSDPI control plane can support a variety of infrastructure types. Further, the KG encryption device prototypes discussed here prove the MSDPI can assure compliance with security policy while enabling a transparent data flow. This report focuses on MSDPI's ability to re-engineer encryption devices, and touches on other DoD/IC requirements such as the development of a new communications test suite and the development of a new routing engine. This report discusses how the MSDPI can easily be exploited to support a wide variety of DoD/IC requirements.

Below is an outline of the specific NRL tasking and related accomplishments for this FY11 project.

Task 1: Install, baseline, and test network equipment provided by the sponsor.
Status: None of the originally planned equipment was delivered, as of the writing of this report.

Task 2: Baseline and test networks between NRL and the partners designated as test bed participants.
Status:

1. Four RHEL5.5 servers and four RHEL5.5 clients were installed in the FEON HSET test bed.
2. Host IB interface connected to IB switches were integrated into the test bed.
3. DDR and QDR initial testing was conducted through QDR Qlogic switch and DDR Longbow switches.
4. QSFP cable compatibility issues were discovered, resolved, and reported.
5. Two IB QDR switches were identified for the FEON effort:
    a. One QDR switch loaned from the Large Data JTCD was integrated into the test bed.

Manuscript approved August 28, 2012.

   b. A second purchased QDR switch was integrated into the test bed.
6. An OpenSIPS SIP server was integrated into the test bed. (SIP = Session Initiation Protocol.)
7. The network management system (NMS) Zenoss was installed and configured, and reported network mapping and interface statistics.
8. Testing was conducted:
   a. Initial OFED perftest suite of tests was conducted and statistics collected.
   b. MSDPI was integrated into the test bed and MSDPI performed point-to-point IB testing, which required performance baselining between QDR/DDR-capable hosts.
9. Modifications were made to the MSDPI test suite:
   a. The point-to-point signaling test command was integrated. This command is the IB send server and client test sequence.
   b. MSDPI Test Master for the test sequence was integrated. This function of the MSDPI allows a tester to issue the test from a remote testing master to a targeted system.
10. The FEON HSET RD&E/T&E test bed was integrated with ATDNet as a core test bed.

Task 3: Develop, test, and incorporate the SIP Domain Discovery Service [2] and MSDPI architecture [1] into the MSDPI prototype where appropriate.
Status:
1. Prototype modifications to the MSDPI SIP Control Plane were completed, which addresses enhanced 10GE/40G IB tests.
2. MSDPI was integrated into the Buildroot 2011 release of the Bay Microsystems operating system.
3. Outstanding tasks were identified for configuring more of MSDPI into the FEON HSET test bed suite of tools and into the Large Data NMS.

2 BACKGROUND

Protecting the confidentiality of data from unwanted disclosure or access is a common requirement for government interests and nongovernment private and public interests. The integrity of the data — guaranteeing that it is the expected data — is also critical to data owners. Finally, assured service is also important: data owners want to be sure the data is available when needed. Common solutions used to provide confidentiality, integrity, and availability of user data make extensive use of cryptography. Typically, this refers to a method in which “plain text” data is made unusable by encrypting it, rendering it illegible to everyone except those parties who can convert the data back to its original, plain text form. Encrypting and decrypting the data is usually accomplished by using a variety of cryptographic techniques and devices. Further, the data is typically protected from unauthorized disclosure by segmenting it into private domains accessible only by known communities of interest. This is commonly accomplished through firewalling, which, simply defined, is a technique that separates protected data from unauthorized access by exercising rules applied to that access. Lastly, data availability is guaranteed by assuring that the systems that host, transport, and protect the data operate free from unwanted control. That is, only those who are assigned to manage the systems can control the systems. The technologies used to protect sensitive data undergo continual revitalization against a continuing and changing threat. This report details a new type of interface for encryption devices that protect networks and sensitive data. Further background can be found in Refs. 2 and 3.

For government networks and private networks to operate securely, new ways must always be found to protect the data traversing the networks. Figure 1 illustrates a network configuration first demonstrated for the IC as early as 1995 [3]. The data’s sensitivity criteria are usually set by the data owner, which may be a government or a private entity. The common method for accessing data, government or private, is through some kind of network access. For example, private sector banking customers conduct remote private financial transactions through the World Wide Web. The typical bank customer is able to conduct

banking business over public networks because transactions are protected by various data-protecting technologies such as Transport Layer Security (TLS) and firewalling.

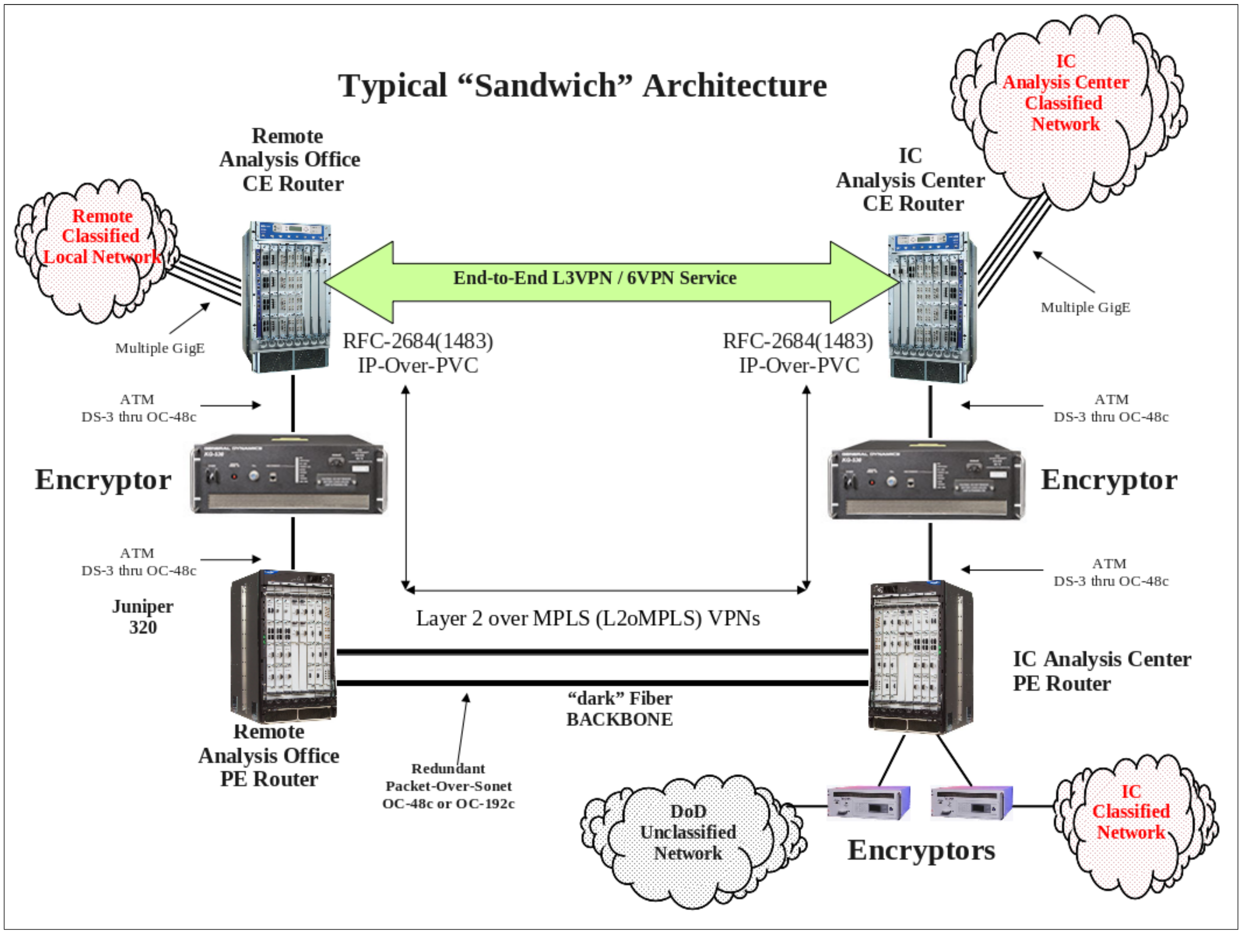


Fig. 1 — Demonstration of an IC control and data plane

However, governments require a higher degree of data protection and therefore make use of cryptographic systems called network encryption devices. Some leading network encryption devices used today are the Asynchronous Transfer Mode (ATM) network encryption device called the FASTLANE [3] and the newer Internet Protocol (IP) network encryption device called the High Assurance Internet Protocol Encryptor (HAIPE, or converted TACLANE).[1] Network encryption devices function as firewalls by segmenting local protected domains from public access, and exploit cryptography algorithms to protect user data during transport between protected domains. Today these devices also include network functionality based on the interfaced network. For example, the FASTLANE not only encrypts/decrypts user data passing through the device, it functions as a limited ATM network switch as well, supporting a

[1] Another type of encryption device commonly used is the link encryption device. This technology is typically used in a point-to-point configuration.

typical suite of ATM network protocols. Even the new HAIPE network encryption devices include network protocols to include the IP and some of the routing protocols found in typical IP network routers.

The process of developing a new network encryption device or even upgrading existing network encryption devices is generally a major undertaking. This is because historically, network encryption devices are an integration of the encryption engine and the network interface; therefore the entire device must be certified and accredited for every network interface even if only one component is modified.

3 SCOPE

It is not the intent of this report to detail every functional requirement of an encryption device. For example, device access control, cryptographic methods, or key management are not covered (it is assumed that existing methods will be employed). Only new traffic and policy management technology is demonstrated. The examples presented in this report are not all-encompassing designs but rather “proof of concept” experiments to explore the capabilities of the MSDPI. It is assumed that further research and development must be conducted to provide an operational deployable system. This report can serve as a guide for a final implementation of commercial products.

4 EXISTING PROTECTION LIMITATIONS OF PROTECTED DATA

Public networks such as the Internet have a less stringent requirement to protect user data than the U.S. government does.[2] Public networks can therefore easily provide policy services such as Quality of Service (QoS). Public networks can integrate protection technologies that are not as constraining as those implemented by government organizations. However, U.S. government organizations such as the Department of Defense and the intelligence community are presented with the additional challenge of providing traffic policy while at the same time implementing restrictive protection technologies. Implementing policy can be a challenge because the exchange of information between the protected and the public domains is typically prohibited by the local information security policy and current techniques. Further, network encryption devices usually are designed to support a specific network to assure availability and performance. For example, the HAIPE can protect IP networks but is limited when used to protect a Frame Relay network. The FASTLANE ATM network encryption device functions well in an ATM network but cannot interface directly to an IP network.[3] Because these devices have one specific network interface, deployment changes and upgrade costs for these devices are generally high (even if the device can support multiple networks, it requires a hardware/software change). Operating these devices may require specific training, deployed architectures may require unique configurations, and logistics can be challenging. Thus, operating these devices becomes an art, as configurations become a balance of driving up protection mechanisms versus network functionality. All these issues promote constant changes with encryption technology. MSDPI reduces the number of required changes to the entire encryption device, helping to reduce costs and logistics. More important, with MSDPI, deployment schedules can be dramatically reduced because encryption devices can be molded to networks without the need to go through the entire operational approval process.

[2] One can argue that financial organizations have as much concern for data protection as does the U.S. government. However, for the scope of this discussion, the concern for data protection is limited to national interests only, that is, protection of national secrets and human lives.

[3] This is not to say the FASTLANE cannot be used to protect an IP network or HAIPEs cannot protect an ATM network. As Ref. 3 points out, the KG-75A is perfectly capable of protecting IP networks. However, doing so requires exploiting the concept of the “Sandwich” architecture detailed in the reference.

5 THE MULTI-SERVICE DOMAIN PROTECTING INTERFACE

One of the features of the Multi-Service Domain Protecting Interface demonstrated in this report is an encryption system that separates the cryptographic function (which must be certified through an expensive and time-consuming procedure) from the network interface function (which does not have to be certified unless it is embedded in the device). The MSDPI demonstrates a network encryption device that provides a seamless interface between the plain text domain (PTD) and cypher text domain (CTD) to ensure that policy is fully supported. The MSDPI defines a technology that functions as an interface between the network and the encryption engine, while still ensuring the two basic functions of a network encryption device separating plain text (PT) from cypher text (CT) through a certifiable encryption engine. The experiments reported here show how the MSDPI is multi-service by supporting both policy control and security. The experiments demonstrate how the MSDPI supports a broad range of data flow types and quality service contracts between peering domains and service provider domains while still maintaining security policy of the user. The MSDPI uses a configurable technique to determine the network structure and adapts to that technology at runtime. Simply put, MSDPI "wraps" the encryption engine with the network technology in use at the time. This wrapping function is transparent to both the encryption engine and the network. These experiments also touch on the MSDPI's ability to adapt to multiple network technologies dynamically. For example, by using the additional service capabilities built into its architecture, the MSDPI simulated interfacing the encryption engine to an IP MultiProtocol Label Switch (MPLS) traffic flow, then switched to a virtual local area network (VLAN) traffic flow, an InfiniBand traffic flow, and an Ethernet network traffic flow. Each of these types of traffic has a unique characteristic that is exploited by the MSDPI to trigger the point in a traffic flow where the encryption engine cryptographic process begins.

The MSDPI architecture has two components that are used to achieve its goal. First, MSDPI relies on pseudowire emulation (PWE) or similar technologies such as Layer 2 Virtual Private Network (L2VPN).[4] This provides the wrapping function of the data flow. The second key to the MSDPI architecture is the Initiator-Responder function (I-R).[5] I-R manages the data through the encryption engine and provides the control plane function. I-R communicates all control instructions to the encryption engine and network components. I-R communications is accomplished using the Session Initiation Protocol (SIP) for Instant Messaging and Presence Leveraging Extensions (SIMPLE). Further, I-R exploits SIP standard-based signaling, SIP authentication, and SIP policy and validation, securing communications between the I-Rs. By using the I-R design, the encryption engine actions, and therefore the data, are protected from compromise, disclosure, and denial of service.

6 MSDPI SERVICE CAPABILITIES

The SIP technology is used by the MSDPI architecture to improve control plane transport and control. The MSDPI design is able to enhance the mechanism for controlling the actions of the encryption device, the user data traversing a KG, exchange of router Routing Information Base (RIB) databases and test set parameters. This design strengthens security by allowing control messages between local domains, yet assures the separation of the local domain from the public domain while still assuring policy control between the local domain and the public domain.

---

[4] The MSDPI can use any technology; the criterion is simply to traverse from one network edge to another using a known set header tag and information block size.

[5] The names Initiator and Responder come from existing network encryption device terminology denoting which network encryption device, taking on the role of the Initiator, begins the establishment of a security association while the other, playing the role of the Responder, listens for and responds to startup request.

Additionally, since the MSDPI architecture makes extensive use of SIP, the architecture inherits, by default, all the routing and security features of SIP to include authentication, authorization, encryption, and message traffic controls and types. The experiments discussed in this report solidify MSDPI's enhanced capabilities for inter-domain peer establishment and maintainability of inter-domain communications through improved inter-domain control plane communications and improved cross-domain policy synchronization.

7 MSDPI USE OF MPLS

Today, MPLS paths (commonly referred to as label-switched paths, or LSPs) is a popular virtual switching technology used by public network service providers and government agencies. LSPs provide two fundamental benefits, the separation of traffic flows based on policy and cost. For example, LSPs can provide a way to cost-effectively manage Voice Over IP (VoIP) traffic while also protecting the traffic from service denials. For the U.S. government, this technology provides a good solution for transitioning from circuit-based networks to converged IP networks. Further, it allows the government to segment traffic based on policy. Where public networks generally use segmentation for billing and service capabilities, U.S. government organizations such as the DoD and IC focus on segmentation for data protection and nondisclosure. MPLS fits well within DoD/IC deployments because of its ability to segment traffic flows while reducing the complexity of the technology needed to deploy IP infrastructure in much the same way as other virtual circuit-based technologies, such as ATM. Because of these benefits, MPLS is used here as a baseline for describing how the MSDPI architecture functions. Where required, other transport technologies are highlighted. However, MPLS is not a requirement for using the MSDPI within any device.

8 MSDPI DESIGN

8.1 Overview

Figure 2 illustrates the two-plane architecture of a network encryption device based on MSDPI, that is, the control plane and the data plane. By dividing the device into the two planes, user data is protected from exposure, yet policy still can be applied to both the plain text domain and the cypher text domain. Further, since the user data path is established prior to the transmission of user data, there is minimal impact on traffic performance by the control plane. This design exploits two prior technologies detailed in Refs. 1 and 2. The MSDPI architecture is not limited to just the encrypting component but includes all components that provide an interface between the encryption component and the surrounding networks or hosts. Further, this architecture includes any networking component that will provide needed functions for transporting traffic through the encrypting component, as depicted in Fig. 3. The MSDPI interface may exist within a single unit or consist of many integrated units. The core components of the MSDPI are the control plane, data transport input/output (I/O), traffic security flow component, and encryption engine. Additional network components such as MPLS, Ethernet, VLAN, or InfiniBand protocol handling components may or may not reside within the core unit. For example, the MPLS to IP transport mapping function may reside in a typical ISP router and is sometimes referred to as an MPLS label switch router (LSR) with the added MSDPI function within it. Figure 4 illustrates how a network with MSDPI network encryption devices would be engineered.

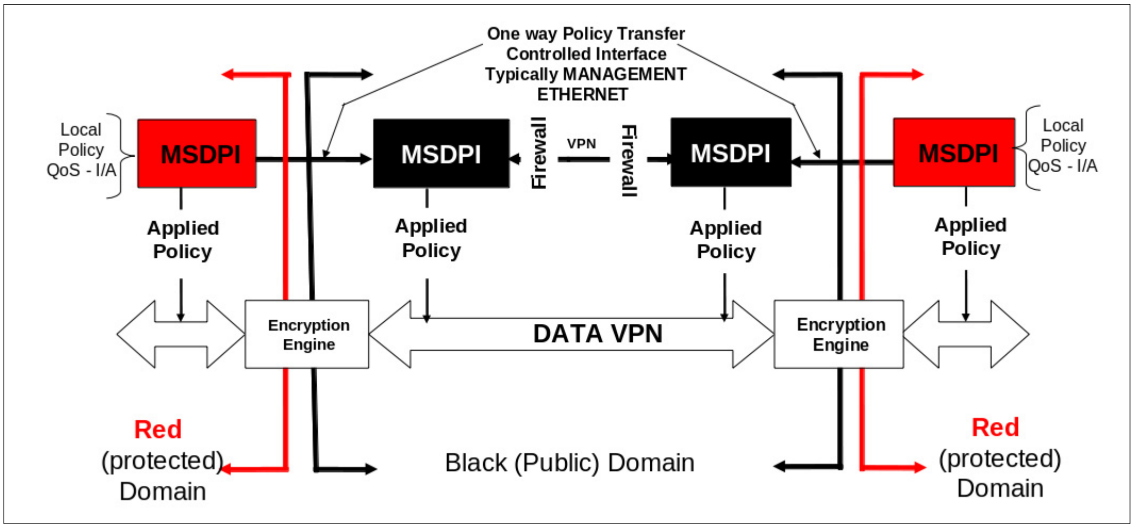


Fig. 2 — MSDPI control plane and data plane

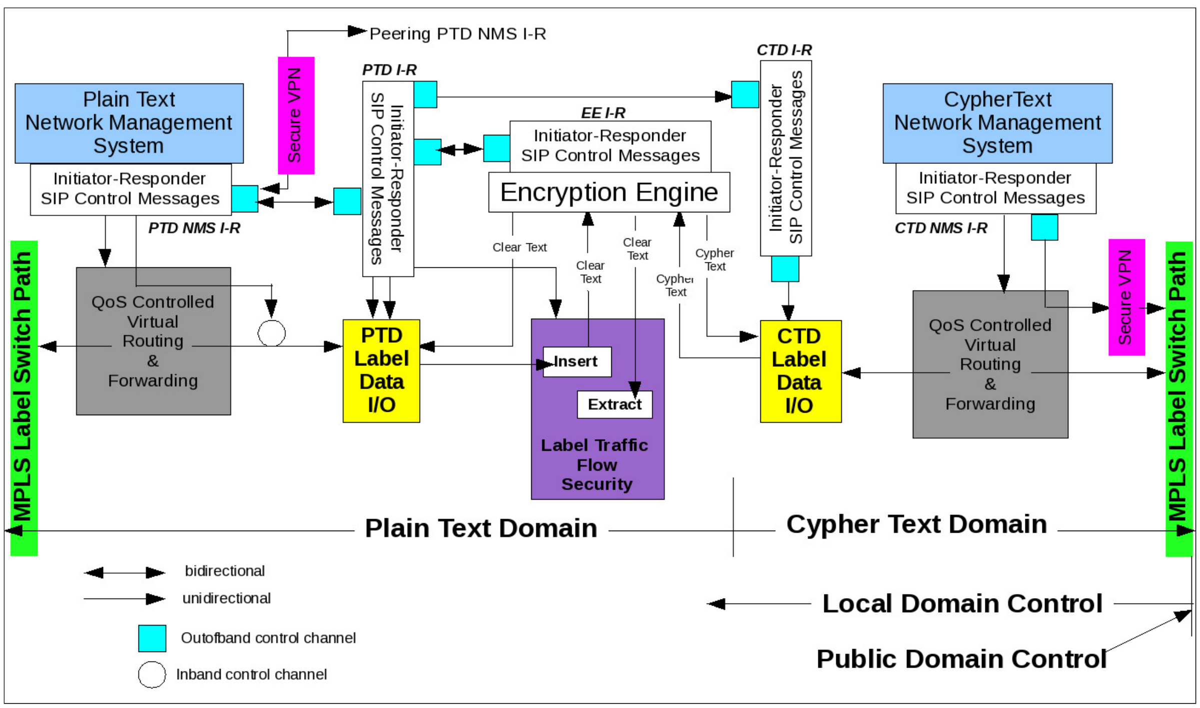


Fig. 3 — MSDPI detailed architecture

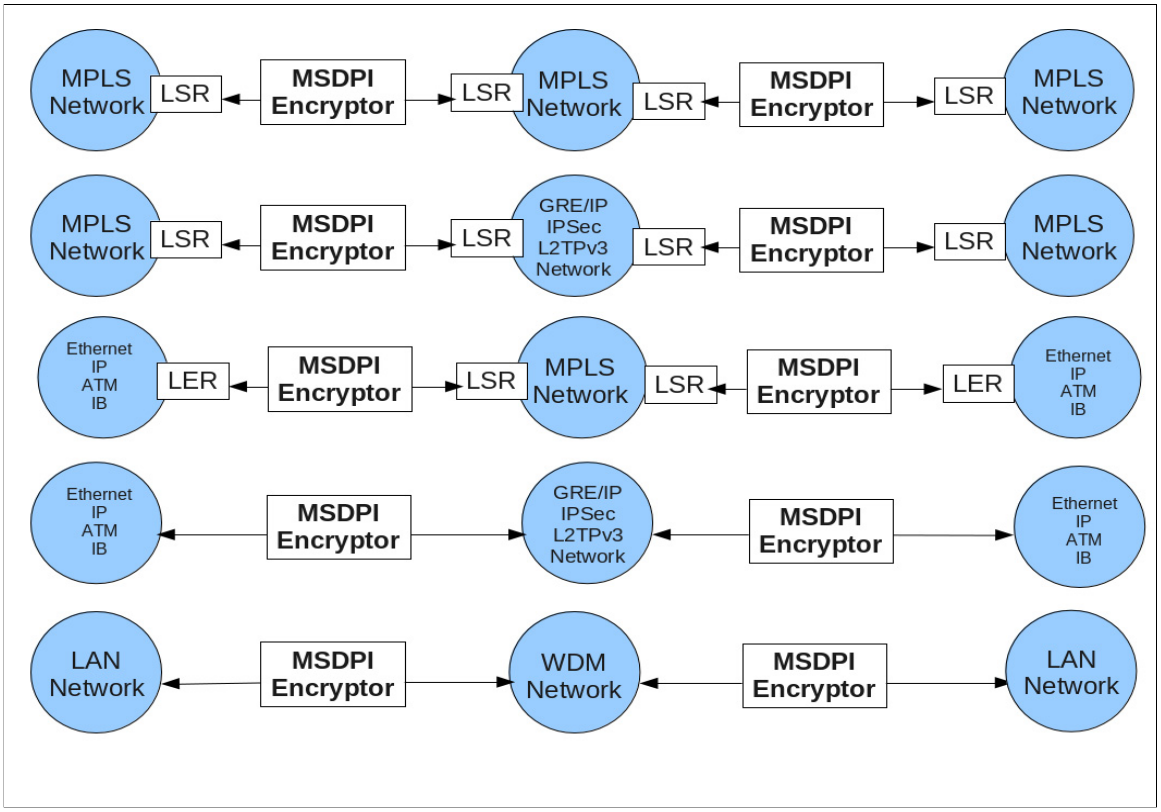


Fig. 4 — MSDPI network interface examples

8.2 SIP Control Plane Design Features

The control plane components experimented with and reported here were introduced in Ref. 1. That publication details how the MSDPI control plane, by exploiting existing SIP technology, specifically SIMPLE technology, was constructed to control various types of traffic flow such as IP traffic. A key component in the interface is the Initiator-Responder. In the MSDPI encryption device architecture, it is used to manage the PTD and CTD data and I/O components and the encryption engine. The boundaries of the network encryption device discussed in this architecture are divided into two specific areas. One boundary is set between the PTD and CTD as denoted at the point which user data is encrypted and decrypted. The second boundary is between the Local Domain Control (LDC) and Public Domain Control (PDC) and is the point at which local administration ends and traffic is controlled by another administrative domain — for example, the handoff of a traffic flow between a local network and a service provider network. As would be expected, LDC consists of all the functions managed by the local domain authority and interfaces to any other private or public infrastructure. The PDC is anything managed by a public network or outside the administrative control of the LDC. Another key distinction with a protected LDC is that the LDC never transports LDC data and/or control information to the public network in an unencrypted format (“in the clear”).

8.3 Network Interface

A feature of the MSDPI network encryption device architecture is the ability to interface to any network infrastructure type, either locally administered or service provided. Figure 4 illustrates some

typical networks this network encryption device will interface to or bridge. The ability to interface to multiple network types is possible because this network encryption device exploits the "Sandwich" technology detailed in Ref. 3 and illustrated in Fig. 1. As Fig. 4 shows, one difference between the configurations is the use of a label edge router (LER). An LER is a network component device that generates or terminates a label switch path; typically it creates a label and prepends or extracts a label from a transport flow. An LER is required when the MLS network encryption device is interfaced to a non-MPLS network, thus the network encryption device is the LSP terminating device for the CTD network. If the network encryption device is to function as a label switch router (LSR), it will interface to MPLS networks through both of its domain interfaces and simply switch labels between the PTD and CTD networks. This is not the case where the network encryption device interfaces to a non-MPLS CTD network, such as an IP backbone network. In this case, the IP backbone interface will simply map the MPLS network encryption device label into the transport mechanism employed. For example, the network encryption device's label would be mapped to an IPSec tunnel across the backbone. Typically, this may involve simply establishing an interface route from the MPLS network encryption device egress port to the IPSec tunnel ingress port.

8.4 Modular Design Features

Another feature of the MSDPI architecture is the modularity of each component. This is made possible because of the I-R design using SIP MESSAGE methods for control plane communications between components. Because each process is controlled by the I-R, the network ingress/egress can be segmented out of the encryption engine (EE). For example, a high-speed router could perform the virtual routing and forwarding (VRF) queuing function and for that matter data input/output function, separating these processes from the encryption process into separate network systems. Synchronizing control functions and controlled traffic flows in this type of architecture is accomplished by the SIP control plane messages. However, the VRF, data I/O, and encrypting engine could very well be housed within a single hardware component. The advantage of a single component is that only one SIP control plane process is required for performing all the control plane processing. The disadvantage is that the encryption engine and network components are integrated into a single hardware component, which will increase development and deployment requirements and possibly require added and/or tuned hardware to meet performance requirements.

8.5 SIP MESSAGE Dialog

8.5.1 SIP MESSAGE Subject Field

The SIP dialog used between the I-R processes is exactly as defined in Ref. 2. As in Ref. 2, the architecture experimented with in the present report exploits the SIMPLE MESSAGE method technology defined in RFC 3438 [4] and therefore has all the SIP functions and controls. The primary SIP SIMPLE MESSAGE method used is the MESSAGE request. The MESSAGE request requires the subject to be a question or answer. The format of the question subject must begin with the key word "Question" and the answer subject line must begin with the key word "Answer." Each of these subject line key words must be followed by a message type and code such that a subject line format is: Question:[type]:[code] or Answer:[type]:[code]. Each of the keys words must be separated by a ":". Spaces are ignored. The "type" key word designates the message type. For example, in Fig. 5 the type key word is "PTDNMSIR" and designates that the message contains Plain Text Domain Network Management System Initiator-Responder (PTD NMS I-R) query-response information. The code key word is a four-character number to further signify the message function. Thus, in Fig. 5, the message is a query message, "Question," for the Plain Text Domain Network Management System Initiator-Responder daemon, given the tag "PTDNMSIR" with the additional command, code "0005," which may designate that the query message

contains a Presence Information Data Format (PIDF) (see Section 8.5.3) with label processing instructions.

```
MESSAGE sip:10.2.1.46:25060 (text/plain)

Frame 303 (1106 bytes on wire, 1106 bytes captured)
Linux cooked capture
Internet Protocol, Src: 10.2.1.10 (10.2.1.10), Dst: 10.2.1.46 (10.2.1.46)
Transmission Control Protocol, Src Port: 43977 (43977), Dst Port: 25060 (25060), Seq: 1249, Ack: 1, Len: 1038
[Reassembled TCP Segments (2286 bytes): #302(1248), #303(1038)]
Session Initiation Protocol
  Request-Line: MESSAGE sip:10.2.1.46:25060 SIP/2.0
    Method: MESSAGE
    [Resent Packet: False]
  Message Header
    Via: SIP/2.0/TCP 10.2.1.10:25062;rport;branch=z9hG4bKB6jvHHSccUDyF
      Transport: TCP
      Sent-by Address: 10.2.1.10
      Sent-by port: 25062
      RPort: rport
      Branch: z9hG4bKB6jvHHSccUDyF
    Max-Forwards: 70
    From: <sip:10.2.1.10:25062>;tag=B6X34Bc86v61F
      SIP from address: sip:10.2.1.10:25062
      SIP tag: B6X34Bc86v61F
    To: <sip:10.2.1.46:25060>
      SIP to address: sip:10.2.1.46:25060
    Call-ID: 82cfda2f-6b9b-122b-69b8-001c2520c5cc
    CSeq: 96587250 MESSAGE
      Sequence Number: 96587250
      Method: MESSAGE
    Subject: Question:PTDNMSIR:0005
    User-Agent: sofia-sip/1.12.6
    Allow: INVITE, ACK, BYE, CANCEL, OPTIONS, PRACK, MESSAGE, SUBSCRIBE, NOTIFY, REFER, UPDATE
    Supported: timer, 100rel
    Content-Type: text/plain
    Content-Length: 1792
  Message Body
```

Plain Text Domain Network Management Initiator – Responder daemon question control plane message

Code to signal a PIDF is present and it contains label information

Fig. 5 — Example of an I-R MESSAGE dialog message

8.5.2 SIP MESSAGE Address Field

One of the power features of the MSDPI is the exploitation of the SIP Uniform Resource Identifier (URI). The typical format of the URI is the form: <scheme name> : <hierarchical part> [? <query>] [# <fragment>] (see RFC 3986 [5]). For MSDPI, the <scheme name> is always "sip". The <hierarchical part> portion of the URI is typically, but does not have to be, in the format "<user@>address:port". Note the field "<user@>" is optional, thus may be omitted. Most of the prototype MSDPI implementations do not require this sub-field. For example, when the MSDPI SIP PTD Discovery Service device is exchanging PTD data, it addresses the data to the target system with the URI in the IP format "sip:<IP>:<port>" or specifically "sip:10.10.10.10:9999". Since the address field is subjective, this field may be an Ethernet MAC address, such as "sip:0a120b340d:9999". It may use an InfiniBand address, such as "sip:0x0002c9030000a60c:9999". Or it may simply be an MPLS label, such as "sip:0001:9999". Therefore, the MSDPI URI format used is dictated by how the MSDPI URI is interfaced to the supporting communications infrastructure.

8.5.3 SIP MESSAGE PIDF Format

As in Ref. 1, the MSDPI architecture discussed in the present report exploits the concept described in the SIP SIMPLE standard and the use of the Presence Information Data Format to exchange SIP

MESSAGE documents.[6] These PIDF documents contain the control plane messages passed between the MSDPI I-Rs. The generic format of the PIDF used within this architecture is illustrated in Fig. 6. As in the SIP SIMPLE PIDF specifications and in Ref. 1, it is suggested that in the MSDPI architecture, the format of the message document be based on the Extensible Markup Language (XML) PIDF format standard. However, if the XML format is used, all tags must start with “<” and end with “>” and a tagged command must be terminated. For example, if a key word is shown as “<TAG>”, then the terminating sequence for this tag must be “</TAG>”. Within the basic PIDF of this architecture the first key word is “<network encryption device>” which identifies the network encryption device. By identifying the network encryption device in the PIDF, the I-R process can be hosted locally and remotely and adds further accountability to the message. Further, this identifier allows one network encryption device to proxy message traffic for another possibly because of policy or access restrictions. The next important tag is “<label>” which is used to signal a payload to the EE. Included in the PIDF is the “<payloadlength>” tag which also is important information for the EE, letting the EE know where a payload terminates. The “<ttl>” identifier allows a time constraint on the attached commands. This prevents lost commands, which have been recovered, from overwriting retransmitted commands. Since the PIDF may hold more than one Initiator-Responder daemon query-respond request, the requests are segmented by the tag “<initiatorresponder>” and each daemon is segmented by the “<identifier>” tag. Figure 6 and Table 1 provide the other default tags found in the MESSAGE request and the function the tag performs. The implementer is free to expand on any of the tags and tag values to best reflect the infrastructure requirements being addressed. The “<checksum>” tag is used to assure message integrity. It is not within the scope of this specification to determine this value; the value used is an implementation concern.

```
"<?xml version='1.0' encoding='UTF-8'?>\n"
     "<Initiator_Responder_PIDF>\n"
        "<encryptor>\n"
         "<encryptor_address>AAA.BBB.CCC.DDD</encryptor_address>\n"
         "<encryptor_ttl>in seconds</encryptor_ttl>\n"
         "<database>\n"
          "<initiatorresponder>\n"
           "<indentifier>[PTD or CTD] NMS/[PTD or CTD] I-R/EE I-R</identifier>\n"
           "<label>[value represented as 0x0000]</label>\n"
           "<labellength>[length of label in bytes]</labellength>\n"
          "<payloadlength>[length of label in bytes]</payloadlength>\n"
           "<type>source/target/push/pop/unidirection/bidirection</type>\n"
           "<action>add/delete/change/enable/disabled</action>\n"
           "<status>startup/ringing/idle/crashed/active</status>\n"
          "</intiatorresponder>\n"
         "</database>\n"
         "<POC>\n"
          "<name>ISSO</name>\n"
          "<phone>telephone number</phone>\n"
         "</POC>\n"
        "</encryptor>\n"
       "<checksum>SHA</checksum>\n"
     "</Initiator_Responder_PIDF>\n"
```

Fig. 6 — Basic I-R PIDF control MESSAGE

6 To get an understanding of how the I-R uses the PIDF, see Ref. 1. The format between the Initiator and the Responder is governed by the specific configuration using the PIDF.

Table 1 — PIDF Tag Definitions

TAG	FUNCTION
network encryption device	Target network encryption device address
network encryption device_ttl	PIDF time to live, when expired PIDF is ignored
database	New database follows
initiatorresponder	New Initiator-Responder control plane section
identifier	Identifies the specific Initiator-Responder
label	Label identifier, signals a bounded traffic flow to the EE
labellength	Length of the label
payloadlength	Length of the payload, this is required by the EE
type	Identifies the label as source, destination, unidirectional, bidirectional
action	How the Initiator-Responder is to process labels
status	Initiator-Responder status
POC name/phone	ISSO contact information
checksum	Checksum or user authentication mechanism

9 MSDPI ARCHITECTURE

9.1 Component Definitions

To help clarify the discussion, we define some additional MSDPI component names shown in Fig. 3. The Initiator-Responder used to control message traffic within the local domain control is called the Plain Text Domain (PTD) Network Management System (NMS) and is referred to as the PTD NMS I-R. The I-R that controls the local domain side Label Data I/O is referred to as the PTD I-R. The I-R controlling the encryption engine is the EE I-R. The CTD Label Data I/O I-R is called the CTD I-R. CTD traffic management and flow is managed and controlled by the CTD NMS I-R. Each of these components is further defined in Section 9.5, Component Description.

9.2 Control Messages

As discussed above, all control message traffic between the I-Rs employs SIP SIMPLE MESSAGE methods. For example, when the PTD NMS I-R is ready to notify the EE I-R that a new traffic flow is to begin, it will build a PIDF document containing the appropriate control information needed by the EE to begin encrypting or decrypting a traffic flow. Figure 7 is an example of a possible PTD NMS I-R SIP SIMPLE MESSAGE PIDF. Table 2 provides a minimum set of PIDF message types and control codes used by the I-R.

```
"<?xml version='1.0' encoding='UTF-8'?>\n"
      "<Initiator_Responder_PIDF>\n"
         "<encryptor>\n"
          "<encryptor_address>AAA.BBB.CCC.DDD</encryptor_address>\n"
          "<encryptor_ttl>in seconds</encryptor_ttl>\n"
          "<database>\n"
           "<initiatorresponder>\n"
            "<indentifier>[PTD or CTD] NMS/[PTD or CTD] I-R/EE I-R</identifier>\n"
            "<label>[value represented as 0x0000]</label>\n"
            “<length>[length of label in bytes]</length>\n"
            “<queuing>[queuing method]</queuing>\n"
            “<QoS_offer>[rate represented as 0x0000]</QoS_offer>\n"
            "<type>source/target/push/pop/unidirection/bidirection</type>\n"
            "<action>add/delete/change/enable/disabled</action>\n"
            "<status>startup/ringing/idle/crashed/active</status>\n"
           "</intiatorresponder>\n"
          "</database>\n"
          "<POC>\n"
           "<name>ISSO</name>\n"
           "<phone>telephone number</phone>\n"
          "</POC>\n"
         "</encryptor>\n"
        "<checksum>SHA</checksum>\n"
      "</Initiator_Responder_PIDF>\n"
```

Fig. 7 — Example of PTD NMS I-R control message

Table 2 — Minimum Set of PTD I-R Type Codes

Message Type	Control Code	Function
PTDNMSIR	0001	PIDF with multiple policy options
PTDIR	0001	PIDF with multiple policy options
EEIR	0001	PIDF with multiple policy options
CTDIR	0001	PIDF with multiple policy options
CTDNMSIR	0001	PIDF with multiple policy options

9.3 Managing Policy Between MSDPI Interfaces

The MSDPI adheres to the concepts developed in Ref. 1 for managing policy. Typical policy agreements might be access controls, authorizations, or QoS. All PIDF exchanges remain within the LDC and peering MSDPIs. Each LDC PTD NMS I-R is responsible for negotiating policy between peering MSDPIs. This includes any change in policy within the LDC that requires a renegotiation between MSDPIs. Figure 8 illustrates this negotiation sequence. Through this design the control plane between MSDPIs is secure. That is, to protect the exposure of PTD policy information from the public domain, only peering MSDPI PTD NMS I-Rs or peering MSDPI CTD NMS I-Rs exchange policy information. Fig. 9 illustrates the architecture of peering MSDPI policy dialog. Specifically, no information is exchanged between the LDC PTD NMS I-R and LDC CTD NMS I-R or LDC PTD NMS I-R and a peer MSDPI CTD NMS I-R. As Fig. 2 and Fig. 9 illustrate, there is no connection between the PTD NMS I-R and the CTD NMS I-R. The assurance of policy between the PTD and CTD is by mapped (configured)

assignment. To illustrate this, the link between PTD MSDPI to peering PTD MSDPI may be divided into several dissimilar policy (such as QoS) transport paths. These transport paths are then mapped (either manually copied or dynamically initialized) to similar CTD MSDPI transport paths. For example, the PTD may have three transport flows with QoS policy assignments of low-priority, best-effort, and guaranteed[7] services. To protect the disclosure of the type of policy assigned to any specific path, network engineers typically then assign a common policy to all the paths between the CTDs (the PDC). For instance, all the CTD paths will be given a guaranteed rate policy. Thus the traffic flow control is established, controlled, and conducted within the protected LDC. The implementation of the mapping is left up to the information security policy administrator. Fig. 10 is an example of the PIDF XML formatted message to control traffic paths and traffic control for a particular path between peering MSDPI PTD NMS I-Rs or CTD NMS I-Rs.

The most difficult challenge for DoD/IC networks is integrating an encryption device into a network while still maintaining policy control. Typically, plain text domains have no easy method to set local policy across or within cypher text domains. Therefore, the usual practice is to separate the PTD and CTD control planes as previously stated. However, the MSDPI does provide a secure method of exchanging policy across dissimilar domains. As illustrated in Fig. 11, policy can be accomplished through the strictly controlled MSPDI protocol. Typically called a "one-way-transfer," MSDPI is suited for this type of configuration because its signaling protocol is specifically formatted, yet the contents can be dynamically set to specific communications needs.

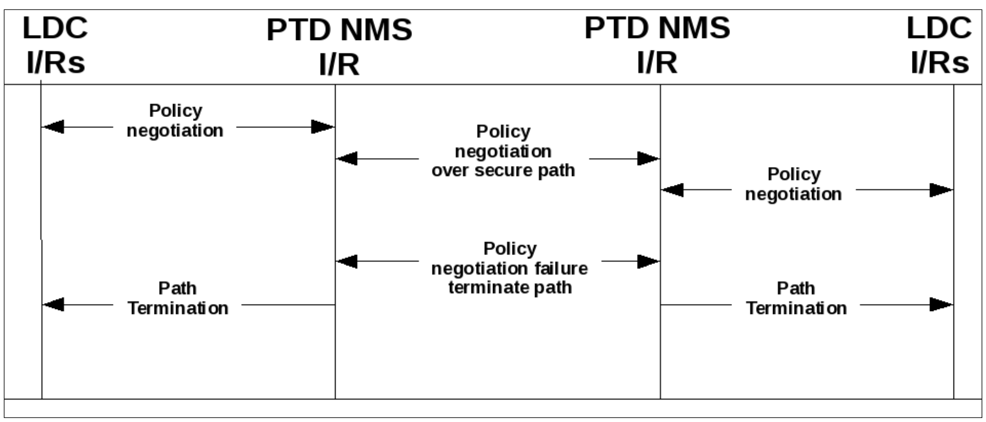


Fig. 8 — PTD NMS I-R to PTD NMS I-R policy negotiation

[7] See RFC 1349 [6] for information on QoS within IP networks.

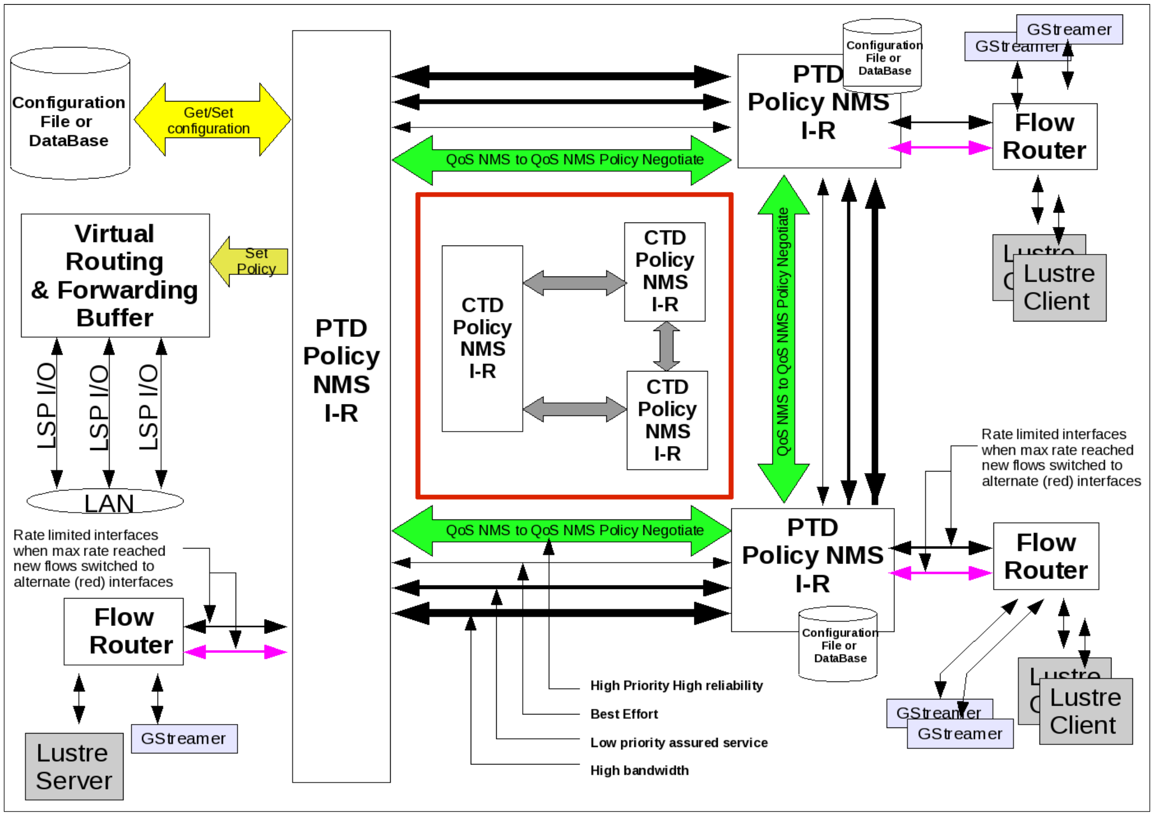


Fig. 9 — Example of peering LDC PTD NMS I-R policy flow

```
"<Initiator_Responder_Traffic_Control>\n"
  "<encryptor>\n"
   "<encryptor_address>AAA.BBB.CCC.DDD</encryptor_address>\n"
   "<encryptor_ttl>in seconds</encryptor_ttl>\n"
   "<database>\n"
    "<initiatorresponder>\n"
     "<indentifier>[PTD or CTD] NMS/[PTD or CTD] I-R/EE I-R</identifier>\n"
     "<labellabel>[value represented as 0x0000]</labellabel>\n"
     "<payloadlength>[value represented as 0x0000]</payloadlength>\n"
     "<type>source/target/push/pop/unidirection/bidirection</type>\n"
     "<action>add/delete/change/enable/disabled</action>\n"
       "<QoS offer>\n"
        "<peer>[IP address]</peer>\n"
        "<rate>[rate represented as 0x0000]</rate>\n"
        "<ctl>\n"
         "<rate>[rate represented as 0x0000]</rate>\n"
         "<remote_port>[remote port number]</remote_port>\n"
         "<local_port>[local port number]</local_port>\n"
        "</ctl>\n"
       "</QoS offer>\n"
       "<tc>\n"
        "<qdisc>\n"
         "<cmd>add/change/replace/link</cmd>\n"
         "<interface>[interface]</interface>\n"
         "<parent>[root/qdisc-id]</parent>\n"
         "<handle>[qdisc-id]</qdisc-id>\n"
         "<parameters>[qdisc specific parameters]</parameters>\n"
        "</qdisc>\n"
        "<class>\n"
         "<cmd>add/change/replace/link</cmd>\n"
         "<interface>[interface]</interface>\n"
         "<parent>[root/qdisc-id]</parent>\n"
         "<handle>[qdisc-id]</qdisc-id>\n"
         "<parameters>[qdisc specific parameters]</parameters>\n"
        "</class>\n"
        "<filter>\n"
         "<cmd>add/change/replace/link</cmd>\n"
         "<interface>[interface]</interface>\n"
         "<parent>[root/qdisc-id]</parent>\n"
         "<protocol>[protocol]</protocol>\n"
         "<prio>[priority]</prio>\n"
         "<filtertype>[filtertype specific parameters]</filtertype>\n"
         "<flowid>[flow-id]</flowid>\n"
        "</filter>\n"
        "<flag>[flag]</flag>\n"
        "<tcstatus>\n"
         "<cmd>[qdisc/class/filter]</cmd>\n"
         "<dev>[interface]</dev>\n"
        "</tcstatus>\n"
       "</tc>\n"
       "<status>startup/ringing/idle/crashed/active</status>\n"
      "</intiatorresponder>\n"
     "</database>\n"
     "<POC>\n"
      "<name>ISSO</name>\n"
      "<phone>telephone number</phone>\n"
     "</POC>\n"
    "</encryptor>\n"
   "<checksum>SHA</checksum>\n"
"</Initiator_Responder_Traffic_Control>\n"
```

Fig. 10 — Example of LDC PTD NMS I-R to peer LDC PTD NMS I-R PIDF

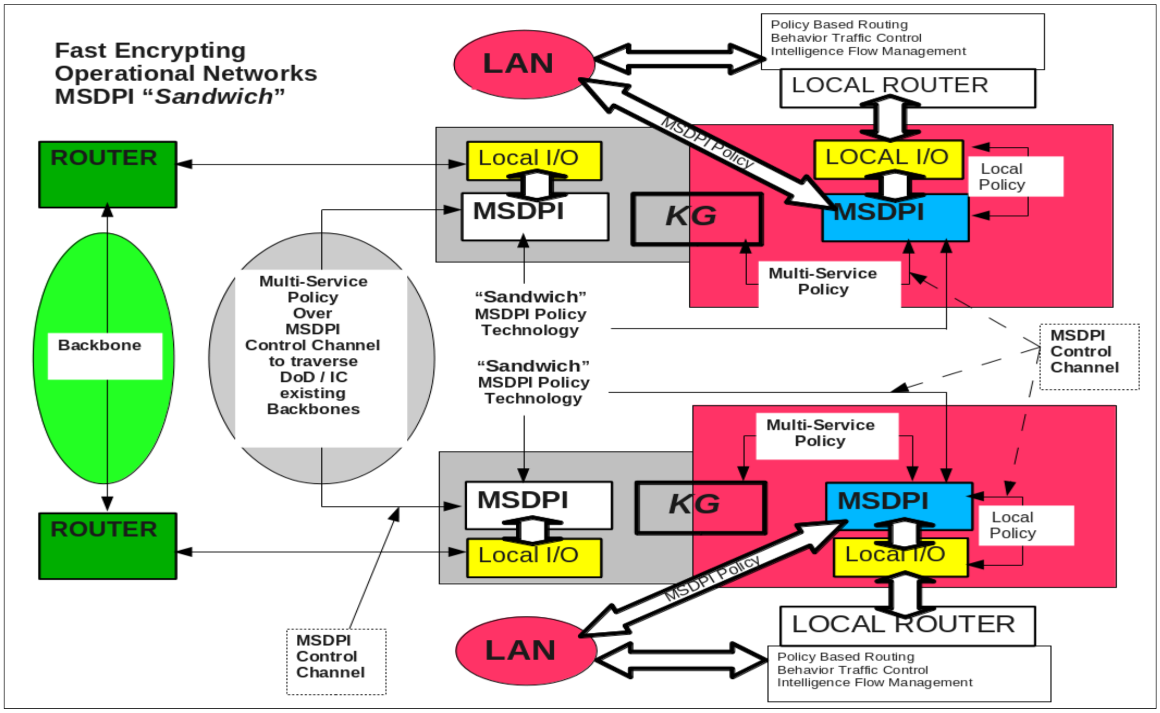


Fig. 11 — MSDPI policy CONOPS

9.4 Encryption Device Peer Discovery

Encryption device peer discovery can be accomplished by either manually setting peer information or by using an automated mechanism. Typically, manual discovery is the simple process in which one network encryption device is statically initialized via a file or database with peer network encryption device security association (SA) information. Automated peer discovery can be accomplished through protocol management and exchange mechanisms. The dynamic peer discovery mechanism that must be used in this design is specified in Refs. 1 and 2.

9.5 Component Description

This section describes each MSDPI I-R daemon and the function it performs.

9.5.1 LDC PTD Network Management System MSDPI Initiator-Responder

The LDC PTD Network Management System MSDPI Initiator-Responder (PTD NMS MSDPI I-R) is possibly the most important component of this architecture. This is because it controls the actions of the virtual routing and forwarding buffers (VRFBs, see Section 11 and Fig. 3), the PTD I-R, and indirectly through the PTD I-R, the PTD Label Data I/O, the EE I-R, the EE, and the CTD I-R and CTD Label Data I/O. Further, the LDC PTD NMS I-R will negotiate policy with peering LDC PTD NMS I-Rs. Figure 3 illustrates the connection architecture between each of the I-R daemons showing how the LDC PTD NMS I-R, as the master daemon, controls all other I-R daemons within the LDC. Figure 12 illustrates the policy negotiation sequence that takes place within the MSDPI. Figure 13 illustrates how the PTD NMS I-R controls the other I-R and processing units (Label Data I/O or EE) either directly or by issuing and

responding to negotiation requests from the other component controlling I-Rs. In other words, when a policy action is required to be executed by the MSDPI, the policy action begins from the LDC PTD NMS I-R which directs the policy action to the LDC PTD I-R. The LDC PTD I-R, in turn, instructs the LDC CTD I-R, which relays that action to its Data I/O component. After the PTD I-R receives the appropriate SIP OK reply, it relays the action request to the LDC EE I-R which, in turn, relays the action to the EE. Finally the LDC PTD I-R passes the policy action to its Label Data I/O and signals the LDC PTD NMS I-R the policy has been established within all the MSDPI I-R daemons. The reason the LDC PTD NMS I-R first issues the requested action to the LDC CTD I-R is to assure the network ingress interface, the CTD Label Data I/O, can handle the policy request. In this way an acceptable policy can be relayed by the MSDPI to the LDC network.

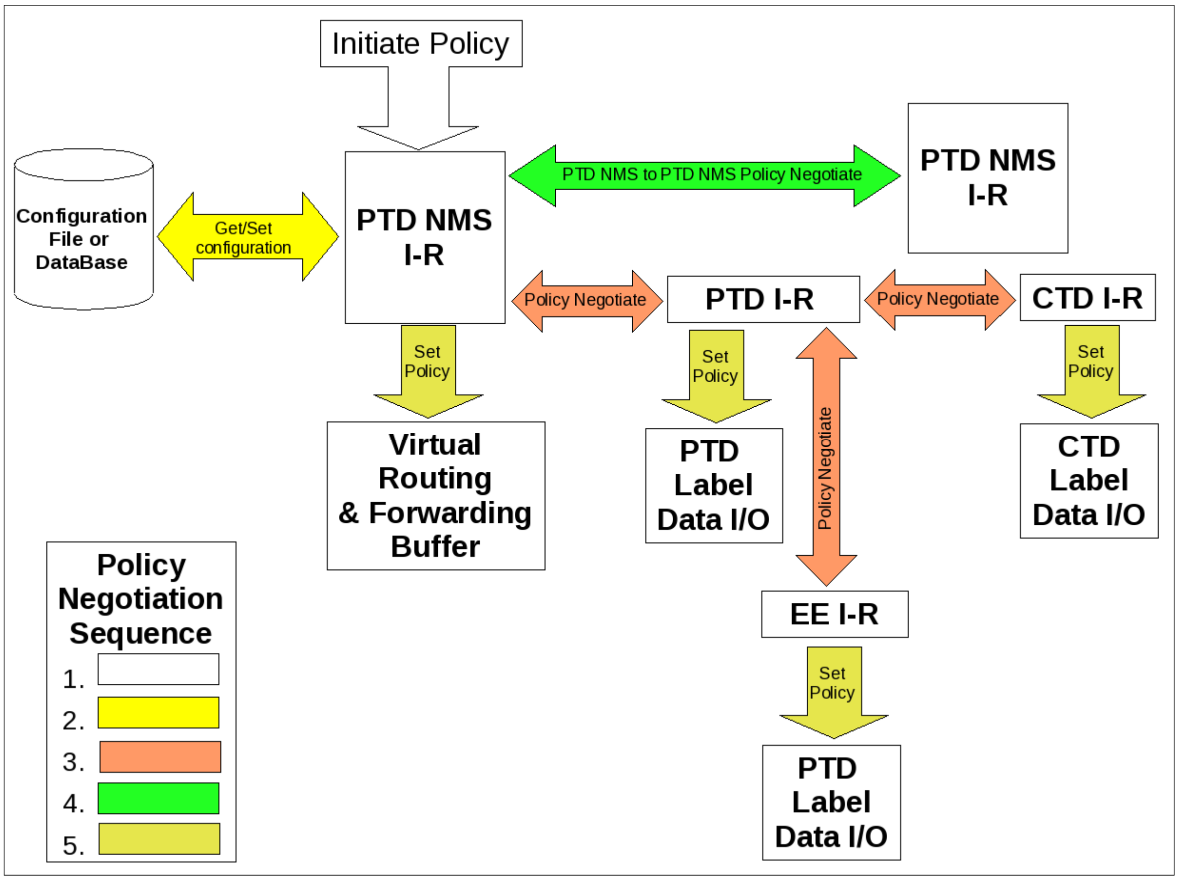


Fig. 12 — Initiator-Responder policy negotiation sequence

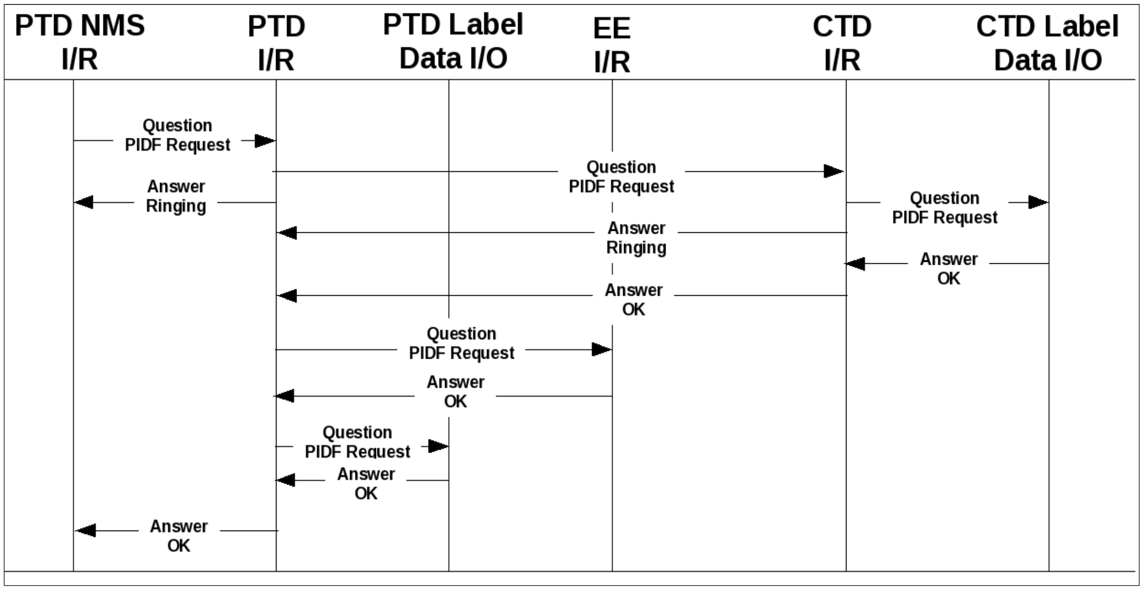


Fig. 13 — LDC I-R command flow

9.5.2 LDC PTD Initiator-Responder

The LDC PTD Initiator-Responder (PTD I-R) controls the actions of the LDC PTD Label Data I/O and the LDC CTD I-R. It issues control commands to the CTD I-R using an I-R MESSAGE PIDF. To provide status and notification feedback, the CTD I-R also uses the MESSAGE PIDF. A typical control key word used in this PIDF is the queuing method employed by the implementer. For example, the queuing method could be Random Early Detection (RED)[8] or even a simple first-in/first-out (FIFO) buffer. SIP MESSAGE control is conveyed from the PTD NMS I-R to the PTD and indirectly to the CTD I-Rs containing the PIDF message illustrated in Fig. 7 and Fig. 9. To assure the PTD Data I/O and the CTD Data I/O queuing mechanisms are synchronized, when the PTD I-R receives the PIDF control message, it issues, as illustrated in Fig. 12, the control message to the CTD I-R. Note the LDC PTD Data I/O will not be signaled with a request until the CTD I/R receives an OK reply from the CTD Label Data I/O to proceed. An example of a proceed message is the “Answer” type code “OK” as illustrated in Fig. 13. This sequence allows the LDC PTD I-R and CTD I-R to negotiate an agreeable policy to assure the LDC PTD request will not overrun the capacity of the MSDPI ingress into the backbone network. Once the MSDPI egress buffer (ingress buffer to the PDC network) control policy is established and the CTD signals the LDC PTD I-R with the “Answer” type code “OK”, the LDC PTD I-R then issues the negotiated command request to its Label Data I/O. When the PTD I-R’s Label Data I/O signals success of this action back to the LDC PTD I-R, the PTD I-R can relay this success back to the LDC PTD NMS I-R. If a successfully policy negotiation between any of the I-Rs cannot be established, resulting in termination of the request, a negotiation failure is reported back to the LDC PTD NMS I-R. This results in the original policy request termination, and the traffic path is not established. At this point the LDC must begin the policy action again. Figure 14 illustrates this sequence of change policy events.

---

[8] Random Early Detection is a congestion control mechanism used to control traffic. There are many such mechanisms. The only concern this design has is providing a mechanism to convey the congestion control mechanism between MSDPI peers.

The PTD (or CTD) I-R controls the action of the Label Data I/O either through the same mechanism just described between the PTD I-R and CTD I-R using the PIDF or through an implementer design. This is because the control of the Label Data I/O is inherently a hardware function typically calling for hardware commands being issued and hardware status queries being processed by the PTD (or CTD) I-R. Again, it should be pointed out, there is nothing stopping the implementer from continually using the PIDF command/request scheme so far discussed as the mechanism used between the PTD I-R and the Label Data I/O MSDPI component.

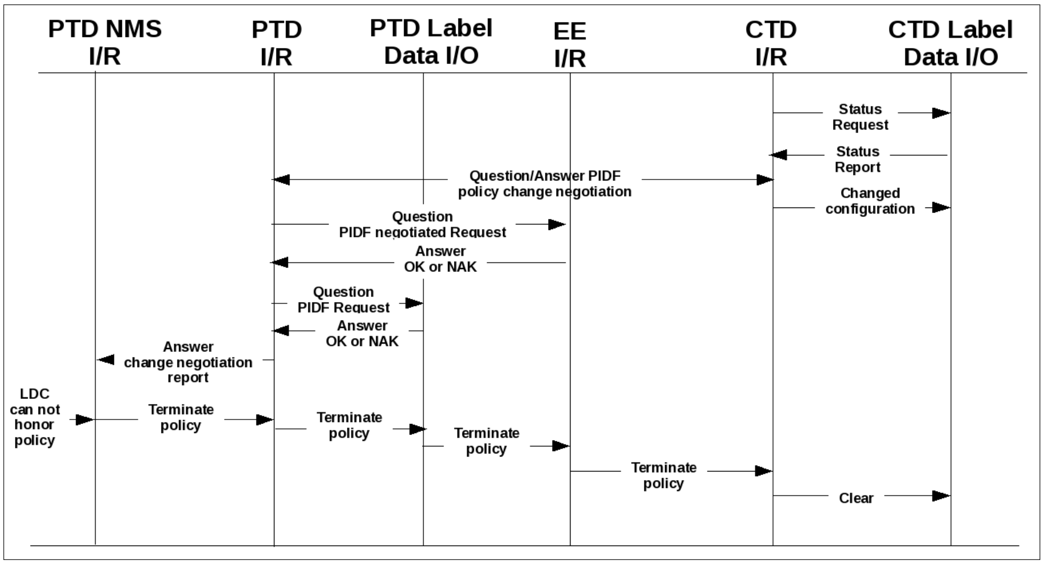


Fig. 14 — LDC I-R policy status/negotiation

9.5.3 *MSDPI Encryption Engine Initiator-Responder*

The Encryption Engine Initiator-Responder controls the actions of the encryption engine. For example, but not necessarily an approved operational control action, the EE I-R may be instructed by the PTD I-R to begin using a specific key management mechanism. Through the dialog between the PTD I-R and the EE I-R as directed by the PTD NMS I-R, the EE will detect and begin the process of cryptographically processing the data traffic. To control flows, the EE will signal the PTD I-R that it is ready to accept traffic and then begin scanning input flows for label information.

9.5.4 *MSDPI CTD Initiator-Responder*

The CTD Initiator-Responder controls the action of the CTD Label Data I/O. It issues control commands to the Label Data I/O using an I-R MESSAGE PIDF message or any mechanism developed by the system implementer. To provide status and notification feedback, the Label Data I/O also uses the MESSAGE PIDF or the mechanism associated to the hardware. To improve security, the CTD I-R has no direct interface to the cypher text domain (PDC); it will not accept any control signaling from the CTD and is only accessible from the PTD and is controlled by the PTD I-R. Figure 3 illustrates this, showing the out-of-band control command arrow going from the PTD I-R to the CTD I-R. Having the PTD I-R issue SIP MESSAGEs to the CTD I-R, which in turns conveys those SIP MESSAGEs to the CTD Label

Data I/O, assures the PTD Data I/O and the CTD Data I/O are synchronized. When a situation warrants that the receiving MSDPI must change the actions of the sending MSDPI, the receiving MSDPI CTD I-R issues a renegotiation request via its PTD NMS I-R to the sending PTD NMS I-R. This change request is sent through the secure control path illustrated in Fig. 3 to the peering PTD NMS I-R. Typically the change will request an adjustment be made to the transmitting VRFBs and/or sending Label Data I/O queues. To signal this request, for example, the CTD I-R builds a MESSAGE PIDF with the key words <action> set to "change" and possibly set the key word <QoS_offer> to some acceptable transmission rate or the key word <queuing> to an acceptable queuing mechanism. This changed policy is then relayed to the peering MSDPI.

9.5.5 CTD Network Management System Initiator-Responder

To provide information assurance (IA), the CTD Network Management System Initiator-Responder is only accessible locally or through an existing security association between managed peering CTD NMS I-Rs. Therefore, peering CTD NMS I-Rs will operate using approved access control and interfaces. For example, peering CTD NMSs will use encrypted authenticated control channels to exchange control information. Figure 3 illustrates this configuration with the out-of-band arrow which connects the CTD NMS I-R to the "Secure VPN" channel. The primary function of the CTD NMS I-R is to manage the LDC CTD virtual routing and forwarding buffer of the MSDPI.

10 CONTROL PLANE

This discussion further details MSDPI control plane messages, components, and control plane protection scheme functionality. Control message traffic and the management of data traffic are reviewed. Further, the discussion details a key component, the MSDPI label, explaining what it is and how it is used. This section provides some details on typical PIDF control messages used between the I-Rs, and the responses for those PIDF messages. Additionally, it describes the scheme used to protect control plane paths.

10.1 Label

To understand how the MSDPI controls traffic flows, the reader needs to understand how the MSDPI uses the protocol header of the traffic traversing the interface. Much like any encryption architecture, the MSDPI takes a flow in from the LDC PTD and directs the EE subsystem to encrypt the flow payload, then forwards the flow to the public network, the PDC CTD. Since the EE can be a standalone component to the entire MSDPI system, the MSDPI needs to bound I/O flows so it can control those flows. It does this by binding the flow to a "label." The MSDPI architecture defines a label as any header information within a traffic flow used to denote the established beginning of a traffic flow. The MSDPI binds the ending of a flow by extracting out the length field of the original header. For example, an MPLS traffic flow will have the packet length extracted from the IP header. This value, adjusted to reflect the MPLS header and MSDI label length, is reported to the EE as the length of the MSDPI labeled flow. The final total length value is inserted into the I-R PIDF tag field <payloadlength>. The implementer is free to develop a manual mechanism for creating the MSDPI label. For example, a database of labels could be used; for traffic flows into the MSDPI, the interface draws labels when needed from that database on a per flow basis. Again, to assure peering MSDPIs are synchronized, the MSDPI assigned as the session Initiator would be required to inform a peering Responder about any labels. Figure 15 illustrates the point where the MPLS header is designated as the MSDPI label thus establishing the beginning of a flow and the point in which the EE is to begin encrypting or decrypting the traffic data.

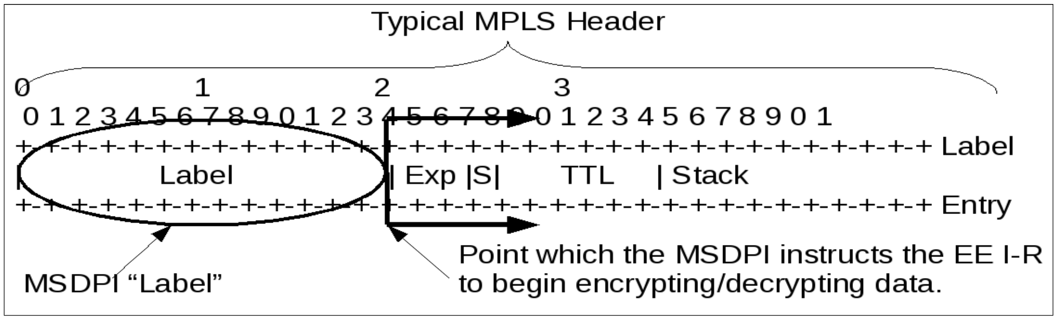


Fig. 15 — MSDPI label encryption

There are other examples of how the traffic header can be transformed into an MSDPI label. The mechanism used can be left up to the implementer. What is important is the label must be consistent between MSDPI peers and operate between different implementations of the MSDPI. Besides MPLS, the Ethernet MAC destination and/or source address (IEEE 802.3), the VLAN tag (IEEE 802.1Q), the InfiniBand Layer 2 local routing header (LRH), and the InfiniBand Layer 3 global routing header (GRH) can be used as an MSDPI label. For example, Fig. 16 and Fig. 17 illustrate, respectively, how the 802.1Q tag and the InifiniBand LRH can be prefixed to the payload traversing the MSDPI so it can be presented to the EE as an MSDPI labeled traffic flow.

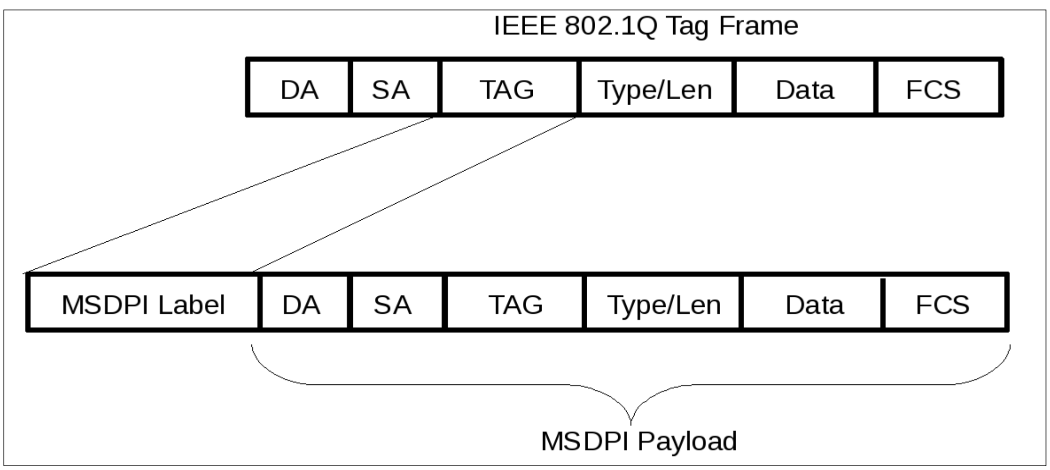


Fig. 16 — 802.1Q MSDPI label

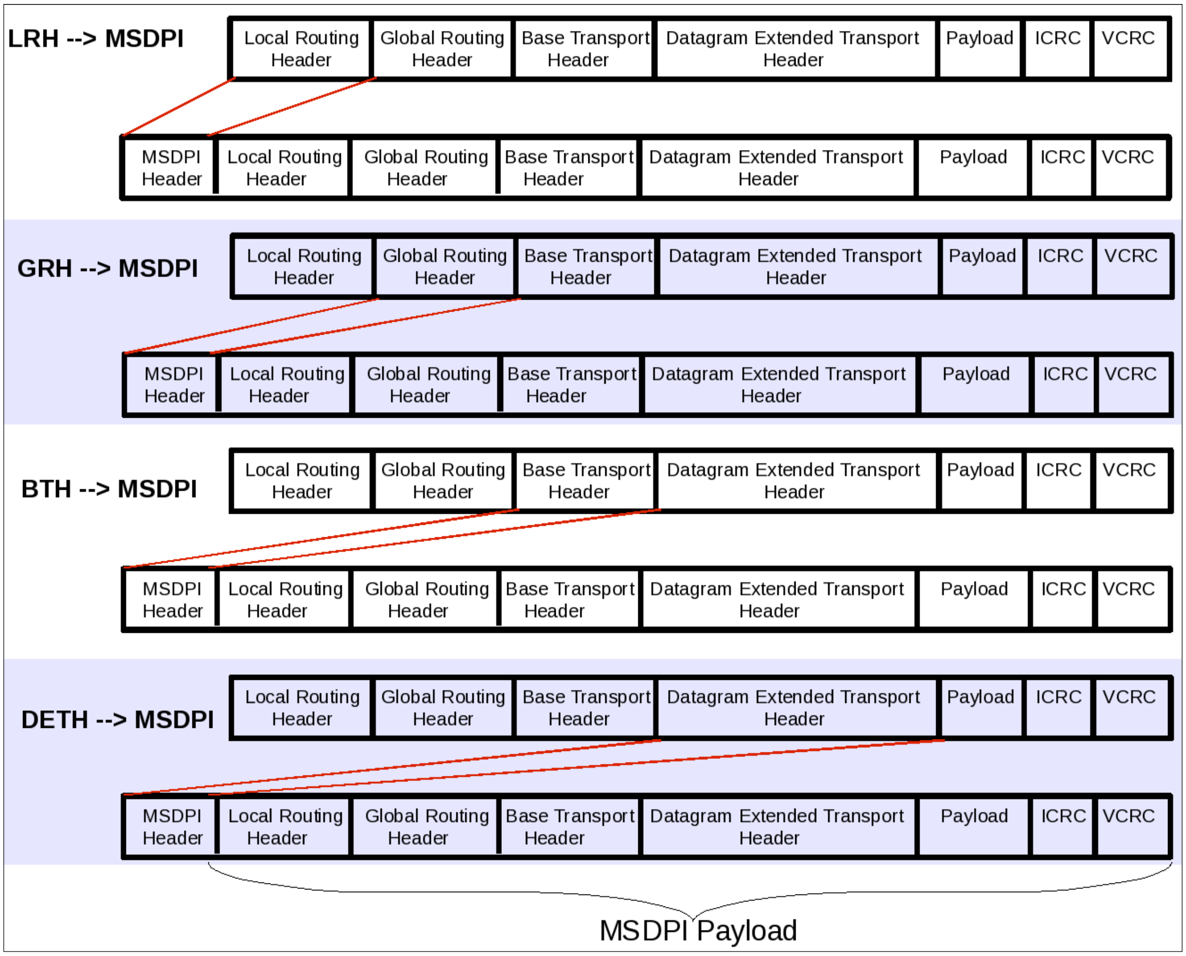


Fig. 17 — InfiniBand to MSDPI label mapping

10.2 Control Channels

The MSDPI can have three types of control channels, an in-band and/or out-of-band control channel. The in-band channel is a transport path established within the same transport path used by the data traffic. Typically the in-band path is segmented into separate paths, one for control traffic and one for data traffic. The segmentation can be, but is not required to be, a separate VPN path between peering MSDPIs. Out-of-band channels are those channels that have physically separate transport paths from the data transport. In either case, the transport path is protected by approved transport security mechanisms such as a prior established protected transport tunnel.[9] It is over these control channels the MSDPI exchanges PIDF SIMPLE MESSAGEs.

[9] The protection of the in-band or out-of-band control channel is out of the scope of this design. It is assumed that approved mechanisms are used to protect control channels. For example, the out-of-band path may be a prior established SSL VPN.

10.3 Example I-R PIDF Messages

As previously stated, the MSDPI exchanges control signaling through the use of a PIDF. The PIDF contains control signals to manage the various subsystems of the network encryption device. For example, the PTD NMS I-R will send a SIP message containing the PIDF illustrated in Fig. 18 or Fig. 19 to signal the beginning of a traffic flow. This PIDF will contain information such as the length of the label, thus providing to the EE the point within the traffic to begin encrypting or decrypting a flow. The following PIDFs are examples of the various control messages. Again, the implementer is free to define what is contained in the PIDF as long as it is an XML format as specified within this design and the format is synchronized between all MSDPIs. In fact, the format can be defined at runtime by a site Information Systems Security Officer (ISSO), providing additional security. The only constraint on the ISSO is that the format must also include basic tags so the PIDF format does not impede the basic functionality of the implemented network encryption device.


```
"<?xml version='1.0' encoding='UTF-8'?>\n"
 "<Initiator_Responder_PIDF>\n"
      "<encryptor>\n"
        "<encryptor_address>10.10.10.10</encryptor_address>\n"
        "<encryptor_ttl>60</encryptor_ttl>\n"
        "<database>\n"
          "<initiatorresponder>\n"
            "<indentifier>PTD NMS I-R</identifier>\n"
            "<label>0x1111</label>\n"
            “<labellength>4</labellength>\n”
            “<payloadlength>1500</payloadlength>\n”
            "<type>bidirection</type>\n"
            "<action>add</action>\n"
            "<status>active</status>\n”
          "</intiatorresponder>\n"
        "</database>\n"
        "<POC>\n"
          "<name>Mr. Security ISSO</name>\n"
          "<phone>111-222-1234</phone>\n"
        "</POC>\n"
      "</encryptor>\n"
    "<checksum>0x1234567890abcdef</checksum>\n"
 "</Initiator_Responder_PIDF>\n"
```


Fig. 18 — Example of PTD NMS I-R to PTD I-R PIDF

```
"<?xml version='1.0' encoding='UTF-8'?>\n"
  "<Initiator_Responder_Traffic_Control>\n"
   "<encryptor>\n"
    "<encryptor_address>10.10.10.10</encryptor_address>\n"
    "<encryptor_ttl>60</encryptor_ttl>\n"
    "<database>\n"
     "<initiatorresponder>\n"
      "<indentifier>PTD NMS I-R</identifier>\n"
      "<label>0x1111</label>\n"
     "<labellength>4</labellength>\n"
      "<payloadlength>1500</payloadlength>\n"
      "<type>bidirection</type>\n"
      "<action>add</action>\n"
      "<QoS offer>\n"
       "<peer>10.10.10.20</peer>\n"
       "<rate>0x200</rate>\n"
       "<ctl>\n"
        "<rate>0x0200</rate>\n"
        "<remote_port>00</remote_port>\n"
        "<local_port>00</local_port>\n"
       "</ctl>\n"
      "<QoS offer>\n"
      "<tc>\n"
       "<qdisc>\n"
        "<cmd>add</cmd>\n"
        "<interface>eth0</interface>\n"
        "<parent>qdisc-id</parent>\n"
        "<handle>[qdisc-id]</qdisc-id>\n"
        "<parameters>[qdisc specific parameters]</parameters>\n"
       "</qdisc>\n"
       "<class>\n"
        "<cmd>add/change/replace/link</cmd>\n"
        "<interface>[interface]</interface>\n"
        "<parent>[root/qdisc-id]</parent>\n"
        "<handle>[qdisc-id]</qdisc-id>\n"
        "<parameters>[qdisc specific parameters]</parameters>\n"
       "</class>\n"
       "<filter>\n"
        "<cmd>add/change/replace/link</cmd>\n"
        "<interface>[interface]</interface>\n"
        "<parent>[root/qdisc-id]</parent>\n"
        "<protocol>[protocol]</protocol>\n"
        "<prio>[priority]</prio>\n"
        "<filtertype>[filtertype specific parameters]</filtertype>\n"
        "<flowid>[flow-id]</flowid>\n"
       "</filter>\n"
       "<flag>[flag]</flag>\n"
       "<tcstatus>\n"
        "<cmd>[qdisc/class/filter]</cmd>\n"
        "<dev>[interface]</dev>\n"
       "</tcstatus>\n"
      "</tc>\n"
      "<status>startup/ringing/idle/crashed/active</status>\n"
     "</intiatorresponder>\n"
    "</database>\n"
    "<POC>\n"
     "<name>ISSO</name>\n"
     "<phone>telephone number</phone>\n"
    "</POC>\n"
   "</encryptor>\n"
  "<checksum>SHA</checksum>\n"
"</Initiator_Responder_Traffic_Control>\n"
```

Fig. 19 — Example of PTD NMS I-R to VRFB PIDF

10.4 Securing the Control Plane

This MSDPI design secures the control plane by permitting write and read control signaling between the LDC PTD NMS I-R and the PTD I-R, the PTD I-R and the Label Data I/O, and the PTD I-R and the EE I-R, but permits only write control signals from the LDC PTD I-R to the LDC CTD I-R. These write signals contain only MSDPI information validated by the PTD NMS I-R to manage the LDC CTD I-R Label Data I/O. LDC CTD I-R status information is the only information sent to the PTD I-R which allows policy negotiations. Further, no signaling is allowed from the PTD to the CTD other than control signaling over a protected channel between peering MSDPI LDC PTD NMS I-Rs. By exploiting the technology commonly referred to as the "Sandwich," detailed in Ref. 2, controlled synchronization of policy between the LDC PTD and the PDC CTD can be accomplished. As in the Sandwich, the MSDPI statically maps control signal behavior of the PTD to the CTD, effectively mirroring, if appropriate, the PTD policy within the CTD. However, it is not required that the CTD exactly mirror the PTD. In fact, the ISSO may determine it necessary to configure the CTD differently from the PTD. The only restriction to implementing the Sandwich is that peering MSDPIs must be configured with equal policy to include the mapping behavior between the LDC and PDC and the mapping between peering MSDPIs.

To further secure the MSDPI control plane, local interfaces exploit an approved transport security mechanism. For example, and not necessarily the mechanism to use, the transport security mechanism used between the PTD NMS I-R and the PTD I-R may be IPSec or TLS. The security method deployed is a matter for the ISO and system accrediting authority. This flexibility to integrate transport security between I-Rs is an additional benefit the MSDPI provides to operational security.

11 DATA PLANE

11.1 Data Flow

The flow of data traffic, as illustrated in Fig. 2 and Fig. 3, flows from the PTD via an MPLS LSP into the MSDPI VRFBs, then through the PTD Label Data I/O, through the Label Traffic Flow Security Insert/Extraction (I/E)[10] to the EE. Then, from the EE, encrypted data flows to the CTD Label Data I/O and finally through the CTD VRFB for controlled injection into the public network. Data traffic from the public network follows the reverse path. It should be noted, the CTD VRFB inbound traffic is typically treated as a simple priority queue if all the buffers have a common policy setting. As stated above, the "Sandwich" architecture relies on the LDC PTD VRFB, and not the LDC CTD VRFB, to control traffic congestion between MSDPIs.

11.2 PTD/CTD MSDPI Virtual Routing and Forwarding Buffers

The MSDPI control of data traffic flows consists of two basic functions in both the PTD and CTD: the control of the data traffic and the data traffic. To control PTD traffic flow, the MSDPI uses the concept of VRFBs as discussed in Ref. 2. The MSDPI VRFBs function similar to how L3VPN[11] service uses VRFs. That is, for L3VPNs, VRFs are used for routing; in the MSDPI case, a policy is assigned to a specific VRFB. Therefore, traffic flowing through the MSDPI is controlled by the VRFB policy assignments. Typically, the PTD VRFBs will have a policy assigned to a PTD MPLS LSP that is originating from the PTD. Also, each CTD VRFB will be assigned to a CTD MPLS LSP. Then by mapping a PTD label to a specific policy and associating the label to a CTD LSP, traffic flows between MSDPIs can be controlled. This mapping may be accomplished by port-to-label, queue-to-label, or even an implementer-specific mechanism. The method used is up to the implementer of the MSDPI. For example, early implementers of the MPLS draft-Martini standard often mapped VCI/VPIs to an MPLS LSP configured physical port. Thus any ATM traffic directed to leave via the port was wrapped into a preassigned MPLS labeled IP packet, effectively mapping the VCI/PCI to a label, ergo a label switch path.

11.3 PTD/CTD Label Data I/O

The PTD and CTD Data I/Os are the interface buffers which accept and transmit data traffic through the encryption engine. The type of buffer queuing used is left up to the implementer. At a minimum, it is assumed that a simple FIFO priority queuing mechanism will be deployed. The primary function of the Label Data I/O is to manage congestion of data through the EE. The queuing mechanism used is not within the scope of this specification and is left up to the implementer.

---

[10] System administrators can determine if the Label Traffic Flow Security I/E needs to be activated in an MSDPI device. It is included here to show where it sits within the data traffic flow through the MSDPI and how it functions as a traffic flow concealing function.

[11] L3VPN, Layer 3 Virtual Private Network, is defined in RFC 2547bis [7].

12 HARDWARE COMPONENTS

12.1 How an MSDPI FPGA Device Functions

Figure 20 illustrates a typical MSDPI programmed into hardware using any commodity field programmable gate array (FPGA) device. For this example, the SIP PTD Discovery Device is used again. Note that the PTD MSDPI communicates with the CTD MSDPI through the SIP MESSAGE PIDF. Using the Q-A dialog, the MSDPIs peer and determine which data streams will be tagged and forwarded.

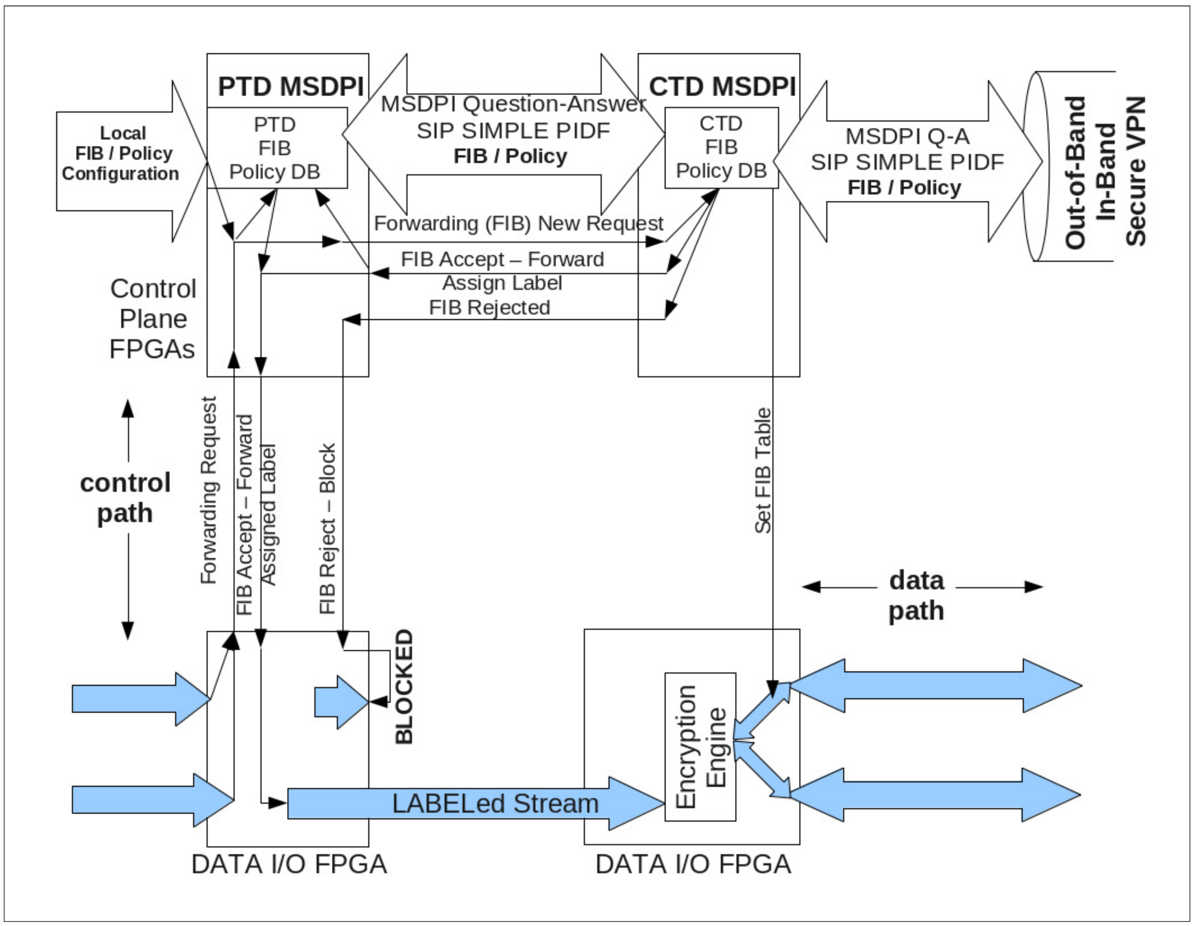


Fig. 20 — MSDPI programmed into hardware

12.2 How an MSDPI FPGA InfiniBand Device Functions

Figure 21 illustrates how the MSDPI FPGA device can be built as an MSDPI-based InfiniBand switch. Since this is an InfiniBand switch, the RIB contains IB LRH and GRH addresses. Note also that the label assignments use the LRH and GRH.

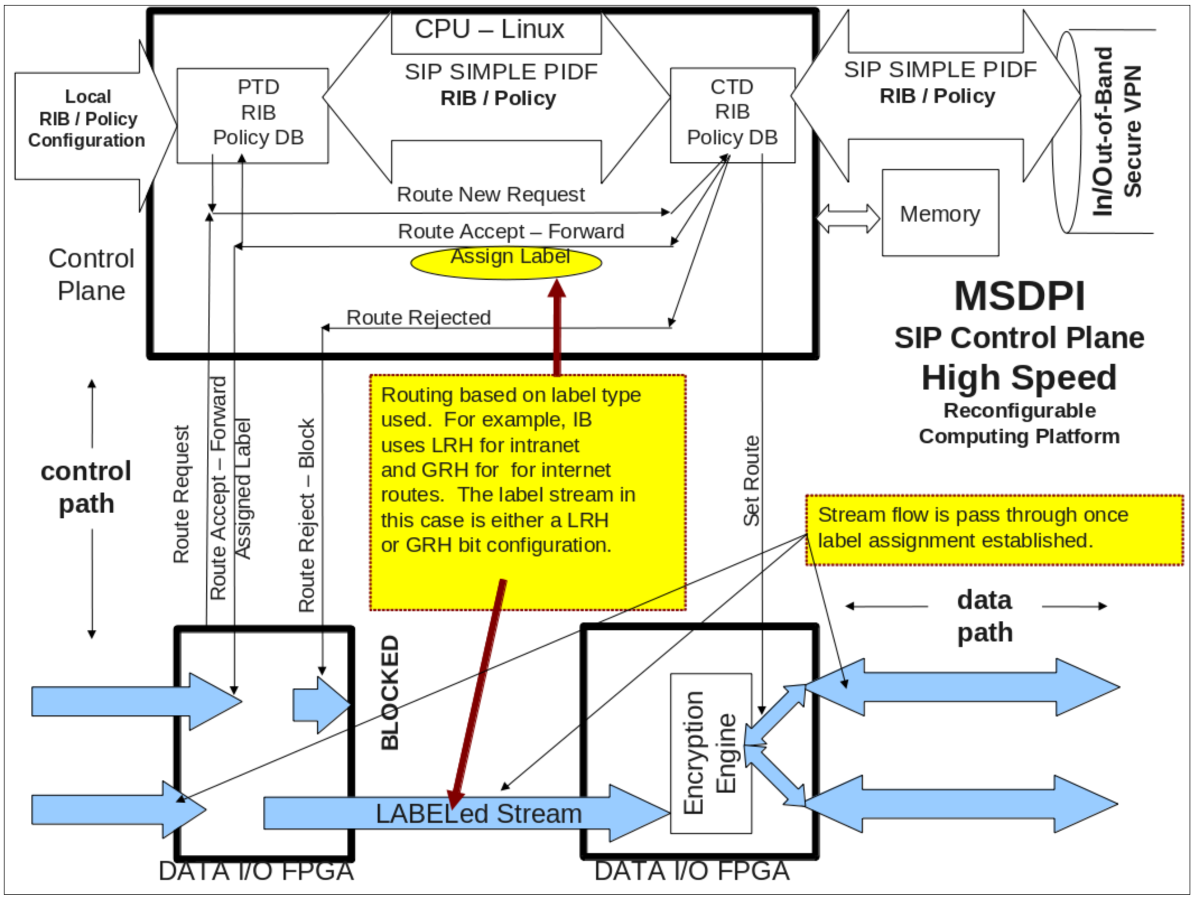


Fig. 21 — MSDPI InfiniBand switch

13 MSDPI CONCEPT OF OPERATIONS

Figure 22 illustrates a typical MSDPI peer-to-peer (P2P) operation and Fig. 23 illustrates a client-to-server operation. Here the reader can see that the concept of operation is exactly like that within a typical SIP session setup with the exchange of SIP "invite" query messages, "ringing" wait messages, and "200 OK" response messages. Part of this setup is the exchange of the PIDF which contains the session's policy information. Once the SIP dialog completes its process, user traffic can begin flowing in accordance with the established policy parameters.

Like any P2P, the MSDPI assumes that P2P configurations are primarily used within local networks. To scale the network, a tiered client-server configuration is suggested, as illustrated in Fig. 23. Therefore,

the SIP dialog between servers is exactly as the dialog between a local client and its local server. When a client requests a session with a peer not within its local server, the server sends invites to its peering servers.

In the client-server configuration illustrated in Fig. 23, a local OpenSIPS server is used to host the various MSDPI clients. Clients register with the MSDPI server using the standard SIP protocol. Then each provides to the server any policy information that pertains to its system's running configuration through the MSDPI protocol. Through the MSDPI protocol, client-server and P2P clients are updated with peer policy data to include tiered servers and meshed clients. This is demonstrated in Fig. 23 which depicts MSDPI signaling exchanges between OpenSIPS servers, the signaling exchange between the KGs through the OpenSIPS proxy, and the signaling exchange between the router and the KG.

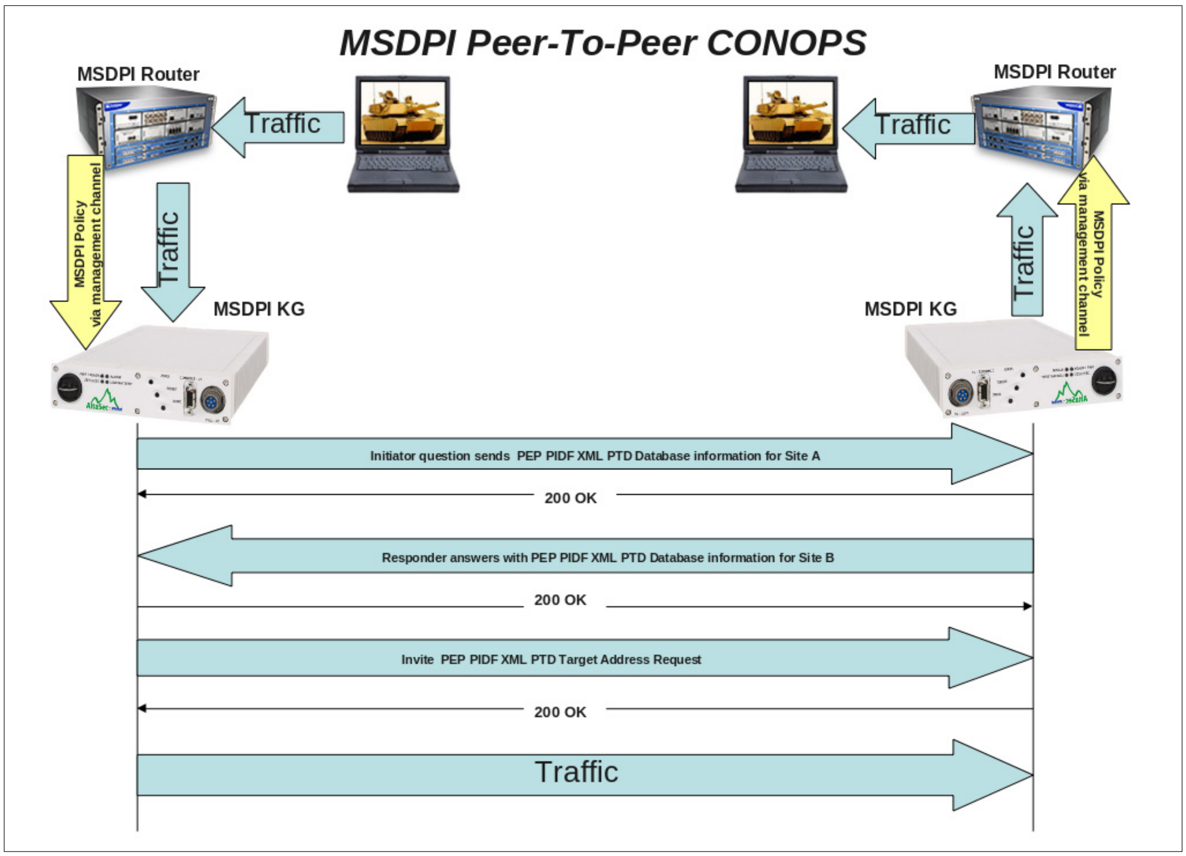


Fig. 22 — MSDPI peer-to-peer (P2P) CONOPS

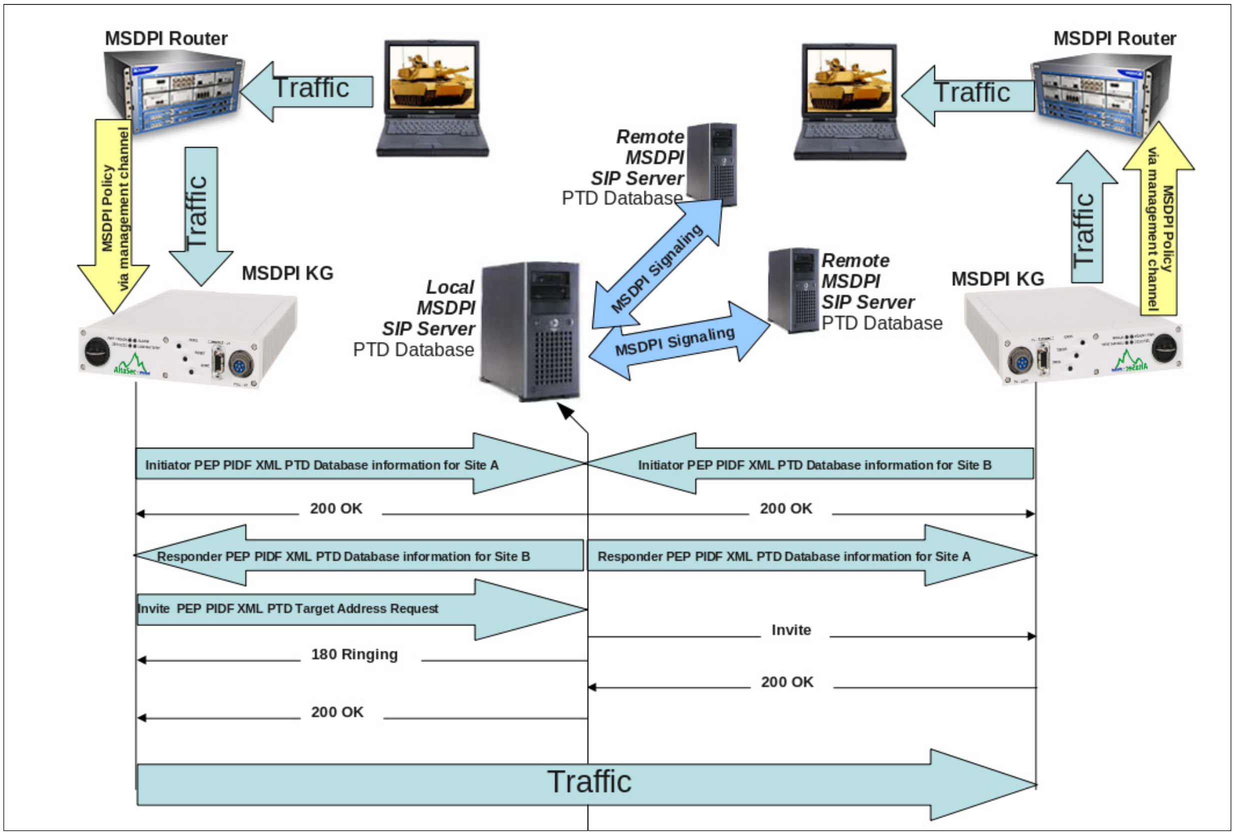


Fig. 23 — MSDPI client-server CONOPS

13.1 MSDPI Warfighter Concept of Operations

Figure 24 demonstrates how the MSDPI would function within a deployed warfighter unit. This capability was proven in the SIP Discovery Service Prototype (Ref. 2, paragraph 16) which simulated a mobile network encryption device exchanging PTD data with a backbone network's network encryption device. Fig. 25 depicts a scenario of a hand-held device using services from a backbone services server as it is switched from one local mobile services server to another mobile server. The MSDPI protocol assures a seamless communications path during this scenario.

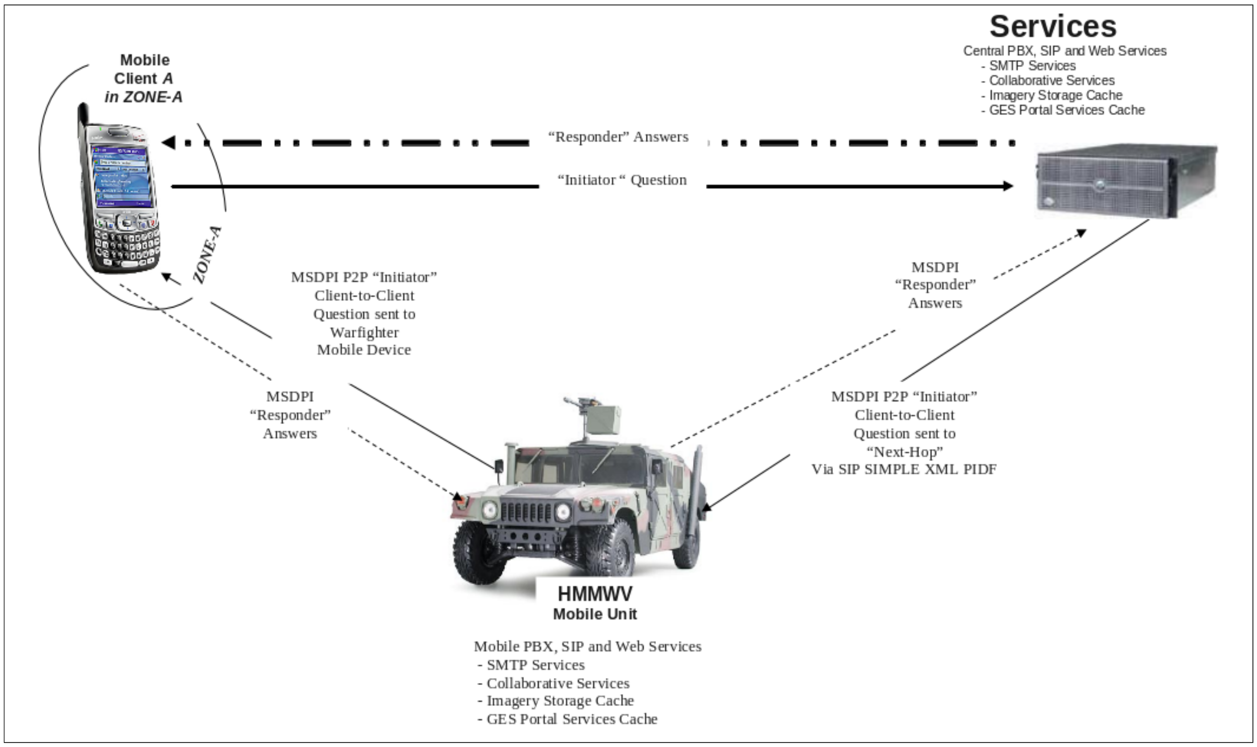


Fig. 24 — MSDPI warfighter CONOPS

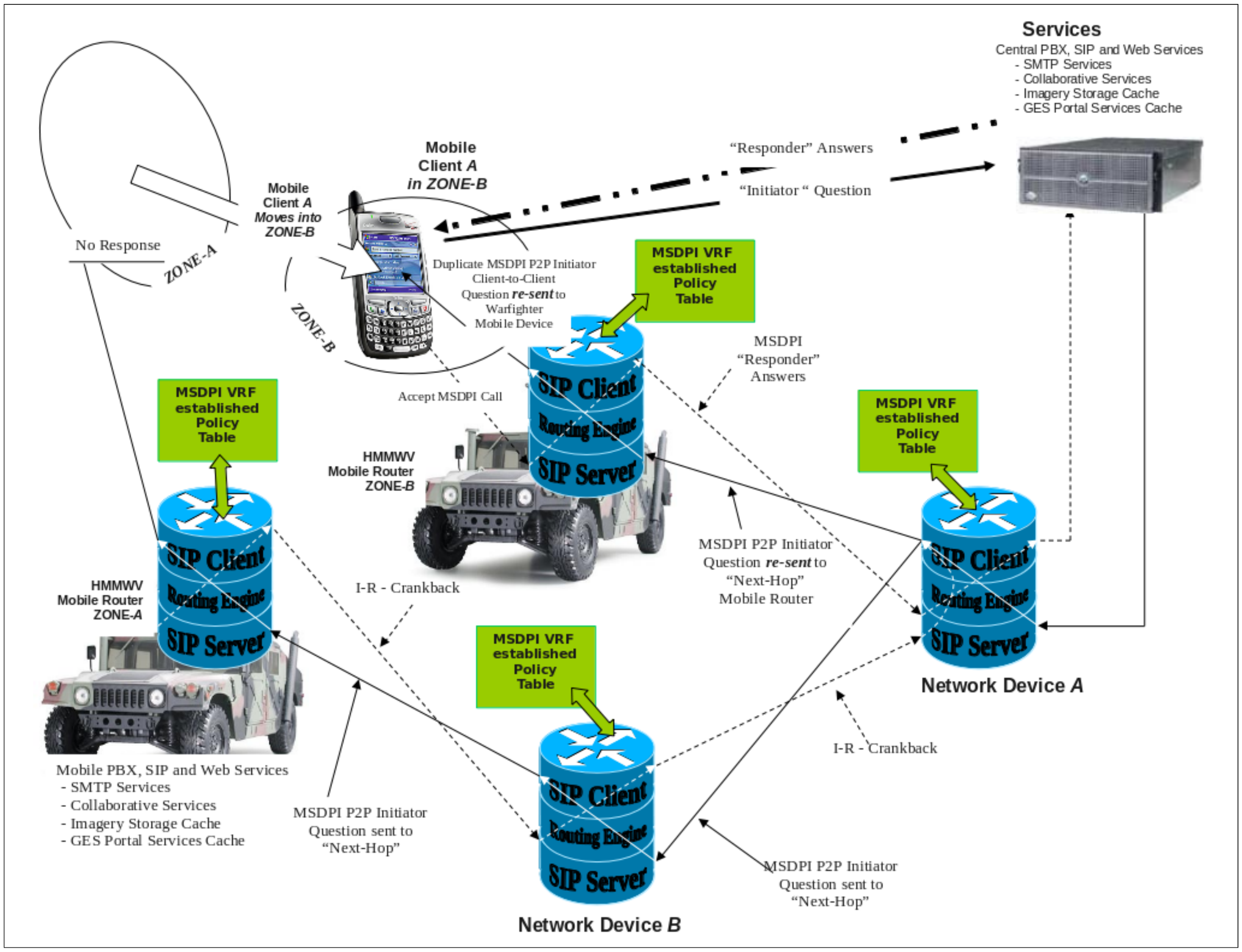


Fig. 25 — MSDPI warfighter reconfigurability CONOPS

14 OPERATIONAL PROTOTYPE EXAMPLES

14.1 Prototype Testing, Architecture, and Commands

14.1.1 How the Prototypes Were Tested

To help demonstrate the concepts discussed in this design, a ping experiment was conducted. Ping is a network diagnostic tool often used to determine the accessibility of one host to another. This experiment included establishing encryption device peers. Note, the key management process and network encryption device logistic deployment and installation were assumed to be established, and only the peer discovery, peer configuration, peer synchronization, and data traffic flow, monitoring, and recovery were demonstrated. Further note, the establishment of communication paths was static. That is to say, a determined set of paths with known service level agreements were assumed. Also, to simplify the experiment, it was assumed that all physical interfaces were initialized to include the secure establishment of the out-of-band control channel between LDC MSDPIs. Additionally, liberties were taken with the subsystems by assuming that several Linux subsystem functions simulated various components of the MSDPI KG. For example, the NMS PTD I-R, PTD Label I/O, and MSDPI control plane interface between these subsystems were simulated by the MSDPI daemon, MPLS-Linux subsystem, and the interface between these Linux components.

The first task in the ping experiment was to establish communications between local LDC MSDPI I-Rs. This was accomplished through the initialization of the PTD NMS I-R which was the first component to run within the MSDPI and configured all the other I-Rs. The next action of the local MSDPI PTD NMS I-R, when configured as an Initiator, was to begin negotiating service level agreements with a Responder. No traffic traversed the MSDPI until the negotiation between MSDPI peers was successfully completed. Finally, traffic policy between peering PTD NMS I-Rs was set during configuration. Then the MSDPI Initiator initialized a security association. The MSDPI configured as a Responder performed any LDC reconfiguration action required to synchronize with the Initiator. Once the policy was established between all the I-Rs — that is, the VRFB assigned policy to egress and ingress label paths per the results of the policy negotiations, the PTD I-R and the CTD I-R configured the Label Data I/Os according to the resulting negotiation configuration instructions, and the peering MSDPIs established communication paths — the ping was sent between systems. The VRFB then began, by a startup command from the PTD NMS I-R, to accept the ping traffic. At that point the ping traffic began flowing from the VRFB to the Label Data I/O and on to the EE, which began scanning the ping traffic, looking for the appropriate label values and the point at which the EE was to begin the encryption process. After being encrypted, the ping traffic was sent to the CTD Label Data I/O for output queuing and then to the CTD VRFB for placing into the appropriate PDC LSP.

14.1.2 MSDPI Prototype Architecture

The prototypes developed for the FEON HSET program consist of three basic components, the Initiator, the Responder, and the Functional Module. Figure 26 illustrates the relationship between the prototype subsystems. For example, the Responder listens on a specified port for incoming SIP SIMPLE MESSAGE traffic. When it receives a SIMPLE MESSAGE, it decodes the message subject line and calls the appropriate Function Module based on the "type" code, passing any attached PIDF data to the Function Module. Upon receiving a request from the Responder to begin processing a SIMPLE MESSAGE, the Functional Module first decodes the command "code" subject line key word so it understands how to process any attached PIDF data. If the SIP subject contains the "Question" key word and the Functional Module has completed processing the incoming request, the Functional Module builds a response SIP subject line containing the "Answer" key word and any appropriate "type" and command "code" key word values. The Initiator is typically only run at system boot-up to initiate any startup dialog

between MSDPIs. For example, it may be used to initialize a security association between peering MSDPI network encryption devices.

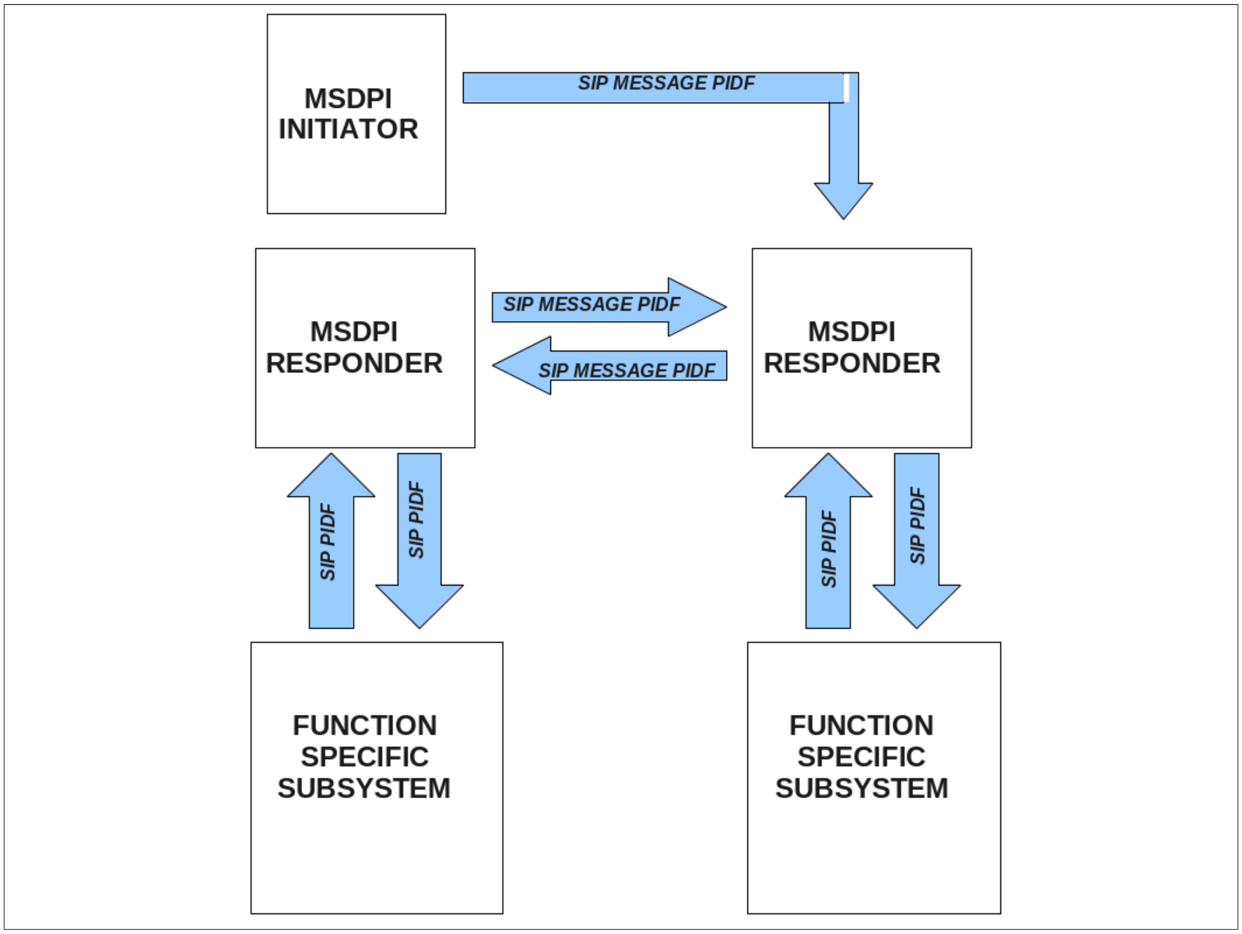


Fig. 26 — MSDPI prototype architecture

Prototypes have been developed as guides for implementing the MSDPI. A prototype was first developed for the SIP-DS [2]. For the current effort, it was enhanced to support a distributed architecture. For example, signaling between subsystems is accomplished through SIP MESSAGE PIDF exchanges translated to Linux Netlink sockets. In fact, because the MSDPI architecture uses SIP MESSAGES for the subsystem component design, the MSDPI can be distributed not only between subsystems within a single operating system (OS), but also between multiple OS and hardware systems. The MSDPI prototypes require several supporting subsystem libraries, such as the "Sofia-SIP" SIP development system, for example. As Fig. 27 illustrates, the MSDPI links directly with the Linux Quagga Routing Information Base subsystem, the Linux MPLS-Linux subsystem, and the Linux OpenVPN and IPSec VLAN services subsystems. Essentially, MSDPI becomes the transport mechanism for these subsystems. For the RIB, LSP, and IPSec databases, updates are accomplished through the MSDPI protocol.

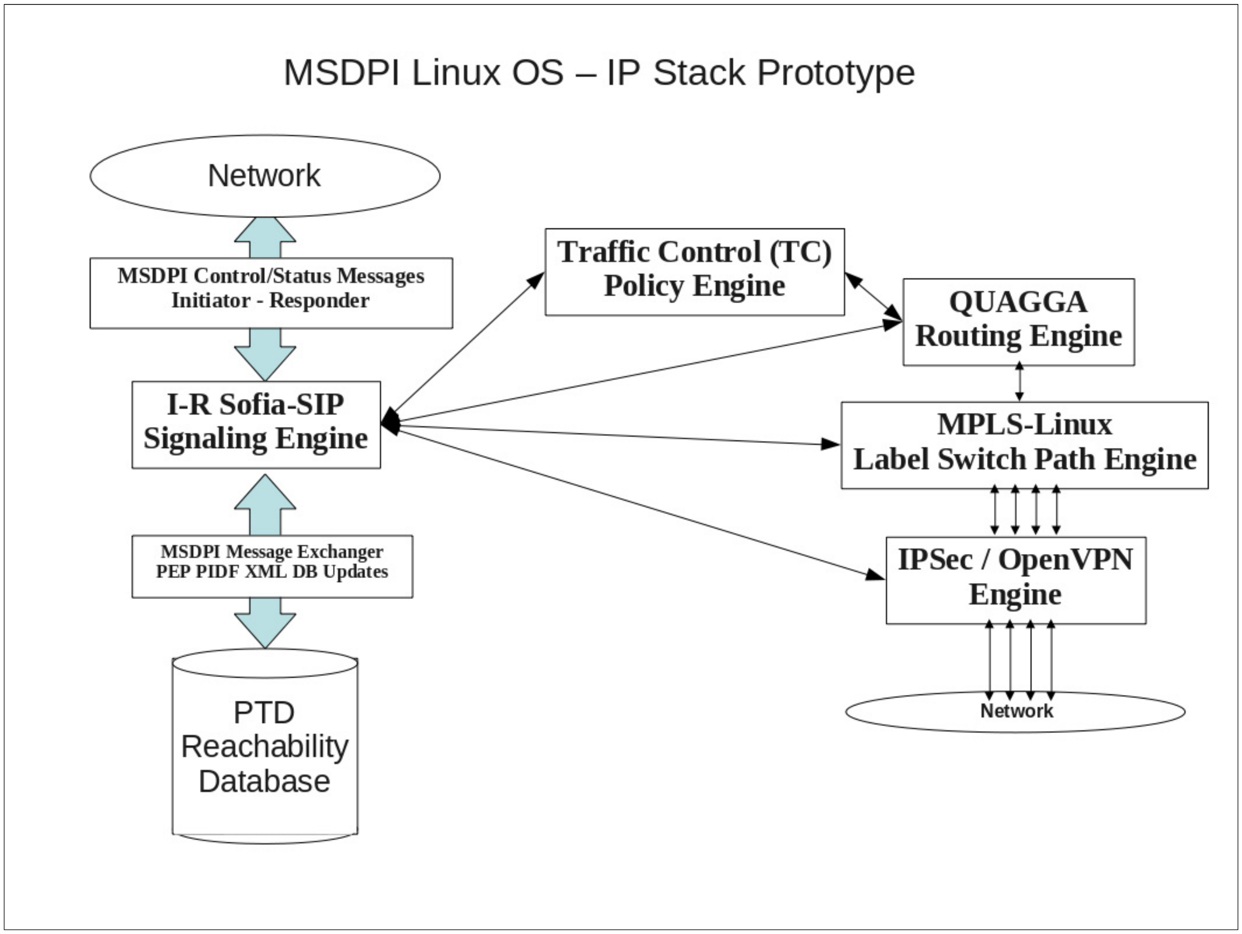


Fig. 27 — MSDPI software architecture

14.1.3 MSDPI Commands

Table 3 lists the MSDPI commands. Because MSDPI exploits the Sofia-SIP application programming interface (API) and its application called “Sofia-cli”, these commands are simply extensions to that application’s command-line. This further demonstrates this architecture’s compliance to existing SIP standards without modification. Basically, the commands are divided into SIP-related commands and MSDPI-related commands. For example, the command “b[ye]” is a typical SIP command and part of the Sofia-cli command set which terminates a SIP client connection with another SIP client or server. “m[essage]” is another typical SIP command, which sends an attached message to a peer.

Some of the more widely used MSDPI commands include “ilptdd” which initializes the local plain text domain database. The “saapplf” command sends an application PIDF to the targeted peer; used by the MSDPI Test Master, this is how each client receives its test directives. Some other MSDPI commands include Route Information Base data updates (how MSDPI performs router address updates), SNMP information updates, IPSec parameter updates (secure VLAN configuration peer updates), plain text

domain database updates to network encryption device peers and, when enabled, other commands such as MPLS configuration parameters and Juniper proxy configuration commands.[12]

Generally, every MSDPI-specific command format is: <command> <URI> <options>. Only the help and list commands exclude the URI and options. Table 3 lists the current MSDPI commands.

Table 3 — MSDPI Commands

COMMAND	**FUNCTION**
addr <my-sip-address-uri>	(set public address)
b	(bye)
c	(cancel)
hold <to-sip-address-uri>	(hold)
i <to-sip-address-uri>	(invite)
k <[method:\”realm\”:user:]password>	(authenticate)
l	(list operations)
m <to-sip-address-uri>	(message)
to <to-sip-address-uri>	(options)
ref <to-sip-address-uri>	(refer)
r [sip-registrar-uri]	(register)
u	(unregister)
p [-]	(publish)
up	(unpublish)
set	(print current settings)
s <to-sip-address-uri>	(subscribe)
llappld <to-sip-address-uri>	(List local in memory APPLication Dbase)
llptdd <to-sip-address-uri>	(List local in memory PTD Dbase)
llscd <to-sip-address-uri>	(List local in memory System Command Dbase)
llsnmpd <to-sip-address-uri>	(List local in memory SNMP Dbase)
lrptdd <to-sip-address-uri>	(List remote in memory PTD Dbase)
lrscd <to-sip-address-uri>	(List remote in memory System Command Dbase)
lrsnmpd <to-sip-address-uri>	(List remote in memory SNMP Dbase)
llIPSt <to-sip-address-uri>	(List local IPSec tunnel information)
clIPSt <to-sip-address-uri>	(clear local IPSec tunnel information)
ilappld <to-sip-address-uri> [action code:0000(load only) or 0001(load & execute)] [local filename]	(Initialize local in memory APPLication Dbase)
ilptdd <to-sip-address-uri> [local filename] [action code:0005(IPSec) or 0006(RIB)]	(Initialize local in memory PTD Dbase)
ilsnmpd <to-sip-address-uri> [local filename]	(Initialize local in memory SNMP Dbase)
irptdd <to-sip-address-uri> [remote filename] [action code:0005(IPSec) or 0006(RIB)]	(Initialize remote in memory PTD Dbase)
ilRIBt <to-sip-address-uri> [local filename]	(Initialize local Route Information Base)
irRIBt <to-sip-address-uri> [remote filename]	(Initialize remote Route Information Base)
ilsc <to-sip-address-uri> [action] [local filename] [actions:0000(init only)0001(init & execute)]	(Initialize local in memory Raw system commandline Dbase)
sqf <to-sip-address-uri> <TAG> <TYPE> <PIDF> <file>	(Send question TAG:TYPE with or without [PIDF file])

[12] These last two commands are not shown in Table 3. To reduce the complexity of the system, some MSDPI commands are built into the system only when needed to support specific environments.

Table 3 (cont.) — MSDPI Commands

COMMAND	**FUNCTION**
saf <to-sip-address-uri> <TAG> <TYPE> <PIDF> <file>	(Send answer TAG:TYPE with or without [PIDF file])
saapplf <to-sip-address-uri> <TAG> <TYPE> <file>	(Answer with APPLication database file: /usr/local/SIPCP/etc/appl.xml)
sqapplf <to-sip-address-uri> <TAG> <TYPE> <file>	(Question with APPLication database file: /usr/local/SIPCP/etc/appl.xml)
saptdf <to-sip-address-uri> <TAG> <TYPE> <file>	(Answer with Plain Text Domain database file: /usr/local/SIPCP/etc/ptd.xml)
sqptdf <to-sip-address-uri> <TAG> <TYPE> <file>	(Question with Plain Text Domain database file: /usr/local/SIPCP/etc/ptd.xml)
sascf <to-sip-address-uri> SYSTEM <TYPE> <file>	(Answer with SYSTEM database file: /usr/local/SIPCP/systemcommand.xml)
sqscf <to-sip-address-uri> SYSTEM <TYPE> <file>	(Question with SYSTEM database file: /usr/local/SIPCP/etc/systemcommand.xml)
sasnmpf <to-sip-address-uri> <TAG> <TYPE> <file>	(Answer with SNMP database file: /usr/local/SIPCP/etc/snmp.xml)
sqsnmpf <to-sip-address-uri> <TAG> <TYPE> <file>	(Question with SNMP database file: /usr/local/SIPCP/etc/snmp.xml)
slptdd <to-sip-address-uri> <TAG> <TYPE>	(Send local in memory PTD Dbase)
slsc <to-sip-address-uri> SYSTEM <TYPE>	(Send local in memory Raw System command Dbase)
slsnmpd <to-sip-address-uri> <TAG> <TYPE>	(Send local in memory SNMP Dbase)
srptdd <to-sip-address-uri> <TAG> <TYPE>	(Send remote in memory PTD Dbase)
srsnmpd <to-sip-address-uri> <TAG> <TYPE>	(Send remote in memory SNMP Dbase)
itcl <to-sip-address-uri> [client list filename]	(Initialize test client list: /usr/local/SIPCP/etc/clientlist.xml)
ltcl <to-sip-address-uri>	(List test client list)
ibt <to-sip-address-uri> [configuration file] <loop count>	(Execute an IB test, file: \"usr/local/SIPCP/etc/ibtest.cf\" contains commandline)
sibpt <to-sip-address-uri> <configuration file> <mode>	(server/client)
stsc <to-sip-address-uri> </usr/local/SIPCP/etc/testscript.xml>	(Send test script)
rtag <to-sip-address-uri> <Subsystem TAG> <IP>:<Port>	(Register subsystem TAG with proxy)
sdi <to-sip-address-uri> <Set default interface IP address>	
ldi <to-sip-address-uri> <list default interface IP address>	
sURI <to-sip-address-uri> <URI [IP:PORT]>	
lURI <to-sip-address-uri> <list URI>	
sst <to-sip-address-uri> <Set System TAG>	(Set this system as either proxy (MSDPI) or subsystem (PTD/TEST/SNMP))
lltags <to-sip-address-uri>	(List local TAGs)
lrtags <to-sip-address-uri>	(List TAGs on URI)
icp <to-sip-address-uri> <0, 1> <filename>	(initialize console printing)
idp <to-sip-address-uri> <0, 1> <filename>	(initialize data printing)
U	(unsubscribe)
z	(zap operation)
info	
e|q|x (exit) <to-sip-address-uri>	
h|?	(help)

14.2 SIP Discovery Service Prototype

Figure 28 illustrates the first prototype test of the SIP PTD Discovery Service (SIP-DS). The SIP-DS, the predecessor and functional equivalent to the MSDPI, is a software daemon that runs in Fedora Linux-based laptops and rack-mounted computers. This prototype test simulated a typical HAIPE KG IPSec[13] VLAN. This exercise demonstrated the SIP-DS (aka MSDPI) simulating the IPSec functionality of a core HAIPE KG and also operating as an edge and mobile HAIPE device. As Fig. 28 depicts, the SIP-DS interconnected IPSec tunnels between simulated KGs. Included in this configuration were connected secure tunnels between backbone KGs and secure tunnels between mobile systems and edge devices. Once all the IPSec tunnels were established by the SIP-DS KGs, traffic was exchanged between the simulated command/analysis center, simulated imagery center, services center, and remote mobile workstations. All SIP-DS XML configurations were built prior to deployment so that the system would boot up into a functional running configuration.

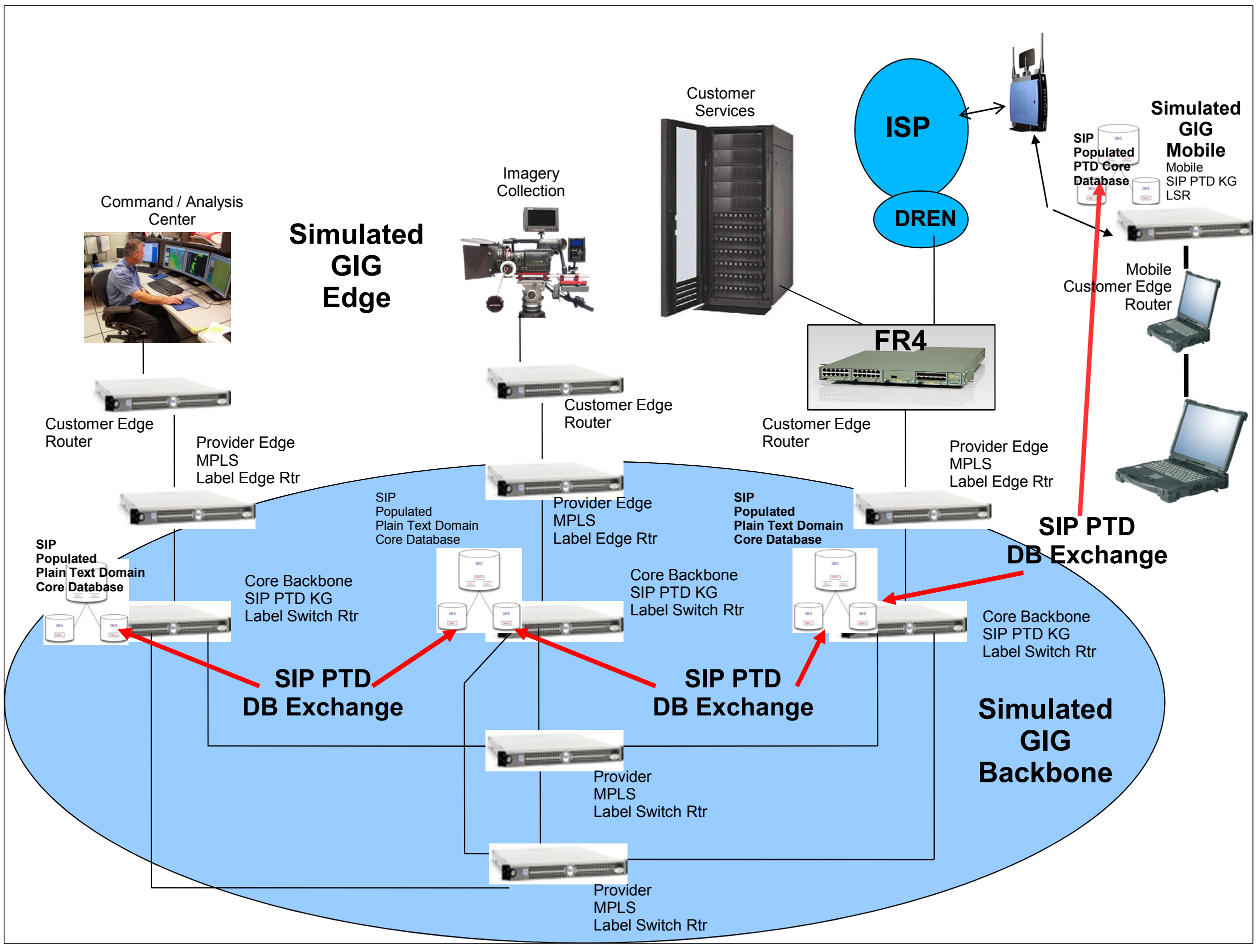


Fig. 28 — SIP-DS prototype architecture

[13] See RFC 2401 [8].

14.3 Commercial Product Prototypes

Figure 29 illustrates how MSDPI integrates with commercial products, using Bay Microsystems components as an example. NRL has demonstrated the viability of running the MSDPI within an over-the-counter network device without having to change the product's development or operational characteristics.

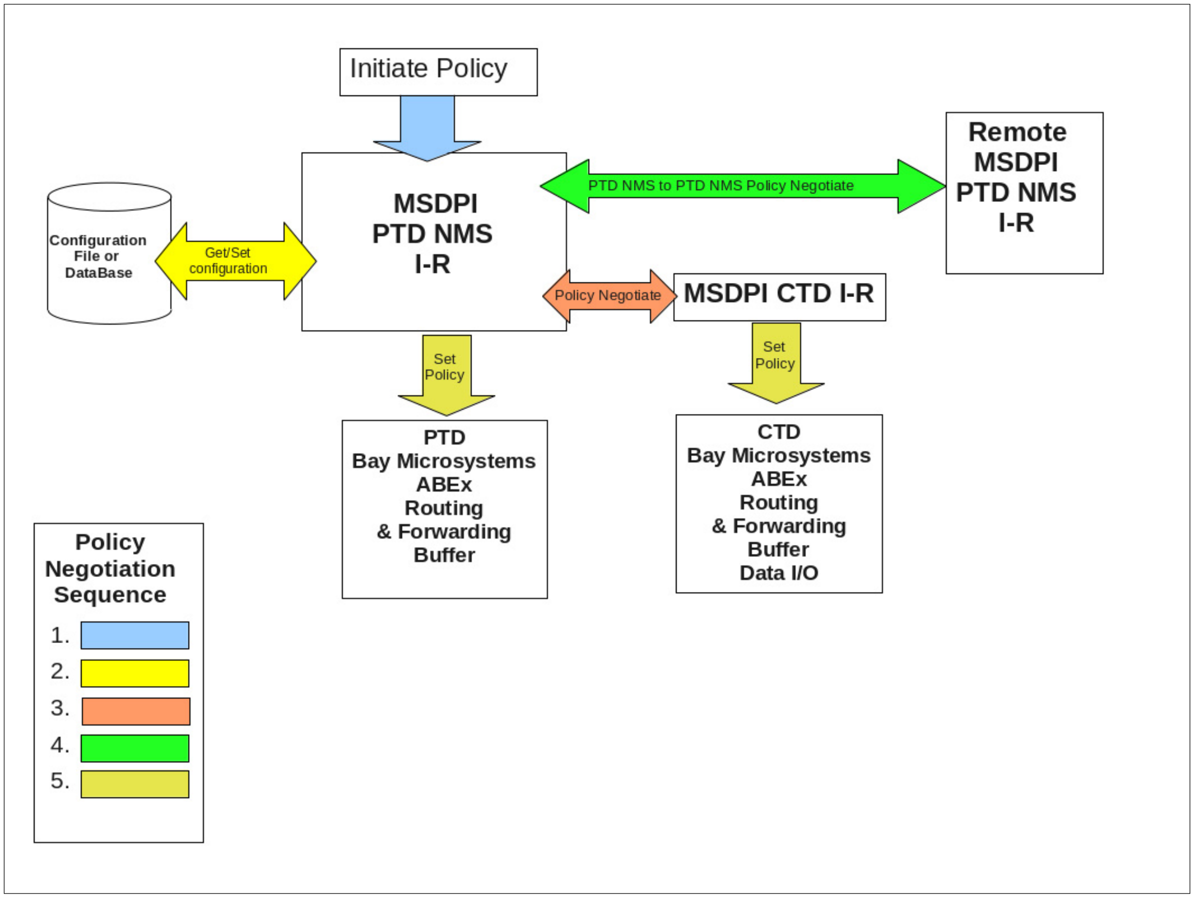


Fig. 29 — Example of an MSDPI commercial prototype

14.4 NRL Commercial Prototype

Figure 30 illustrates the first MSDPI FPGA prototype. The purpose of this development effort was to build a republication system for prospective vendors wishing to learn how to implement the MSDPI. This prototype used the sIXis Reconfigurable FPGA development system called the SY1000-DS. This system consisted of several FPGA components, two designated as "ANDY" and "BARNEY" which processed all system I/O. The processor responsible for all core MSDPI functions included a Xilinx Virtex 5 V5LX1100 called "CHARLIE." The "C" and "D" FPGA components functioned as the EE I-R. Virtex 5 V5LX220 "A" and "B" functioned as the MSDPI NMS PTD/CTD I-R components. "A" and "B" performed policy and label processing. As Fig. 30 shows, all intra/inter-system communication is through the MSDPI protocol.

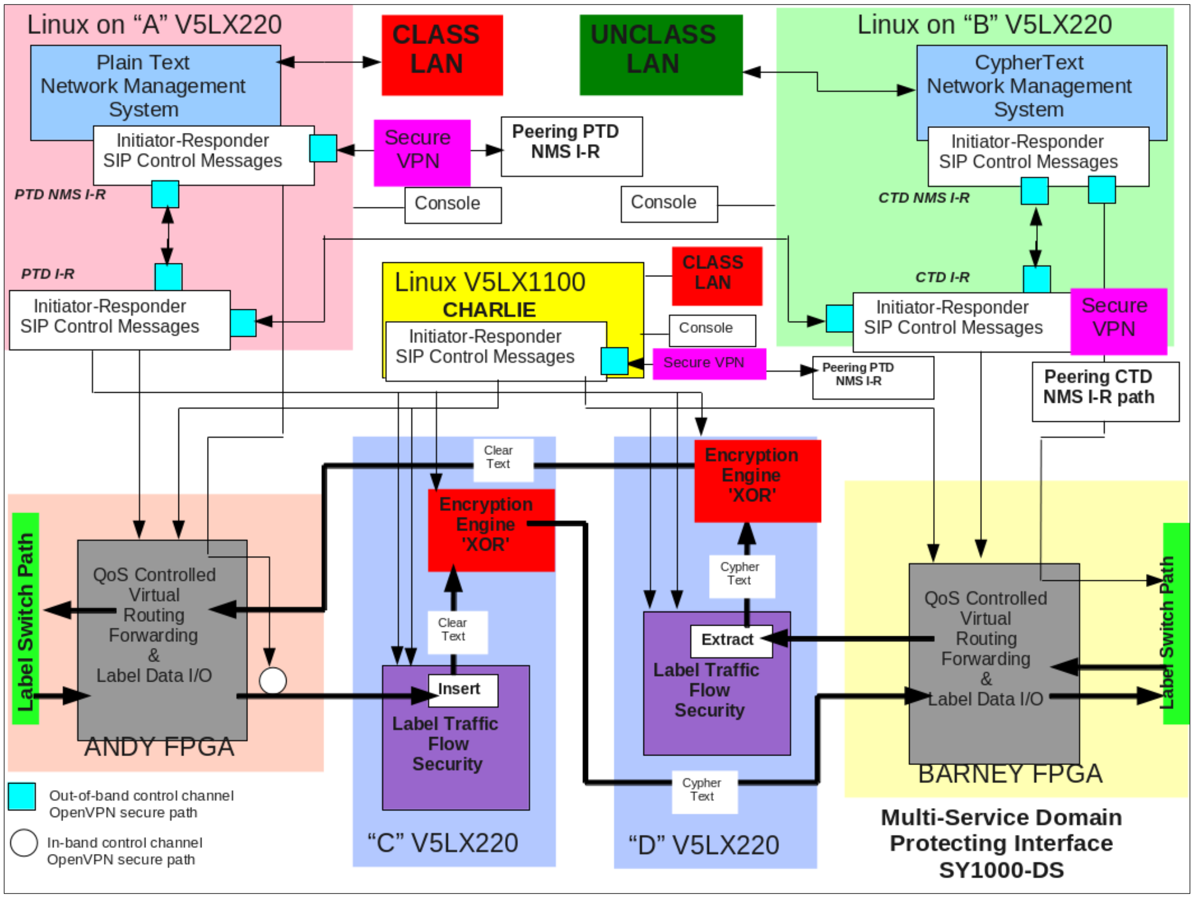


Fig. 30 — First MSDPI FPGA prototype

14.5 Bay Microsystems Product Prototype

14.5.1 Why the Bay Microsystems ABEx and NP10 Network Devices

The Bay Microsystems product line was selected as part of the FEON HSET program for two reasons. First and foremost, Bay was willing to partner with the government for the integration of MSDPI into their ABEx/NP10 network device. Second, like several other commercial products, the Bay product exploits Buildroot.

14.5.2 Buildroot [14]

The Buildroot is a set of Makefiles and patches that makes it easy to generate a complete embedded Linux system. Buildroot can generate any or all of a cross-compilation toolchain, a root filesystem, a kernel image and a bootloader image. Buildroot is useful mainly for people working with small or embedded systems, using various CPU architectures (x86, ARM, MIPS, PowerPC, etc.): it automates the building process of your embedded system and eases the cross-compilation process.

---

14 This section is a direct quote from the Buildroot home page: buildroot.uclibc.org.

The major Buildroot features are:

- Can handle everything in your embedded system development project: cross-compiling toolchain, root filesystem generation, kernel image compilation and bootloader compilation. Buildroot is also sufficiently flexible that it can also be used for only one or several of these steps.
- Is very easy to set up, thanks to its menuconfig, gconfig and xconfig configuration interfaces, familiar to all embedded Linux developers. Building a basic embedded Linux system with Buildroot typically takes 15-30 minutes.
- Supports several hundreds of packages for userspace applications and libraries: X.org stack, Gtk2, Qt, DirectFB, SDL, GStreamer and a large number of network-related and system-related utilities and libraries are supported.
- Supports multiple filesystem types for the root filesystem image: JFFS2, UBIFS, tarballs, romfs, cramfs, squashfs and more.
- Can generate an uClibc cross-compilation toolchain, or re-use your existing glibc, eglibc or uClibc cross-compilation toolchain
- Has a simple structure that makes it easy to understand and extend. It relies only on the well-known Makefile language.

Buildroot is maintained by Peter Korsgaard, and licensed under the GNU GENERAL PUBLIC LICENSE V2 (Or later). Stable releases are delivered every three months.

14.5.3 Incorporating MSDPI into Buildroot for the ABEx/NP10

The following scripts were developed to build MSDPI for operation in the ABEx/NP10. Table 4 is the Buildroot application package script responsible for the cross-compiling of the MSDPI application.

Table 5 is the required Sofia-SIP libraries compilation script. The MSDPI and Sofia-SIP Buildroot menu scripts are found in Table 6 and Table 7. Following Buildroot procedures for adding user-specific applications, the following edits and application-specific scripts were made and added to the Buildroot scripts and build process.

1. The following entries were inserted into the file “Config.in” which resides within the package’s root directory. These changes provide for the selections of the MSDPI and Sofia-SIP application packages.

 - source “package/msdpi/Config.in”
 - source “package/sofiasip/Config.in”

2. The appropriate archive files “msdpi-08Dec10-0800.tar.gz” and “sofiasip-1.12.10.tar.gz” are created and placed in the root directory ./buildroot/dl. Note, the filename of these files must match the names used in the corresponding compilation script.

Table 4 — MSDPI Buildroot Package Compilation Script

```
MSDPI_VERSION:=08Dec10-0800
MSDPI_SOURCE:=msdpi-$(MSDPI_VERSION).tar.gz
MSDPI_SITE:=http://localhost/BUILDROOT
MSDPI_DIR:=$(BUILD_DIR)/msdpi-$(MSDPI_VERSION)
MSDPI_BINARY:=msdpi
MSDPI_TARGET_BINARY:=usr/bin/msdpi
MSDPI_MAKE_OPT = LIBS="-lreadline -lncurses -lpthread -lgobject-2.0 -lgmodule-2.0 -lgthread-2.0 -
lrt -lglib-2.0 -lsofia-sip-ua -lsofia-sip-ua-glib"

$(DL_DIR)/$(MSDPI_SOURCE):
	$(call DOWNLOAD,$(MSDPI_SITE),$(MSDPI_SOURCE))
$(MSDPI_DIR)/.source: $(DL_DIR)/$(MSDPI_SOURCE)
	$(ZCAT) $(DL_DIR)/$(MSDPI_SOURCE) | tar -C $(BUILD_DIR) $(TAR_OPTIONS) -
	touch $@
$(MSDPI_DIR)/.configured: $(MSDPI_DIR)/.source
	(cd $(MSDPI_DIR); rm -rf config.cache; \
		$(TARGET_CONFIGURE_OPTS) \
		$(TARGET_CONFIGURE_ARGS) \
		./configure \
		--target=$(GNU_TARGET_NAME) \
		--host=$(GNU_TARGET_NAME) \
		--build=$(GNU_HOST_NAME) \
		--prefix=/usr \
		--sysconfdir=/etc \
	)
	touch $@
$(MSDPI_DIR)/$(MSDPI_BINARY): $(MSDPI_DIR)/.configured
	$(MAKE) -C $(MSDPI_DIR)
$(TARGET_DIR)/$(MSDPI_TARGET_BINARY): $(MSDPI_DIR)/$(MSDPI_BINARY)
	$(MAKE) DESTDIR=$(TARGET_DIR) -C $(MSDPI_DIR) install-strip
	mkdir -p $(TARGET_DIR)/usr/local
	cp -dpf $(MSDPI_DIR)/src/msdpi $(STAGING_DIR)/usr/bin/msdpi
	cp -dpf $(MSDPI_DIR)/src/msdpi $(TARGET_DIR)/usr/bin/msdpi
	cp -dpfR $(MSDPI_DIR)/SIPCP $(STAGING_DIR)/usr/local
	cp -dpfR $(MSDPI_DIR)/SIPCP $(TARGET_DIR)/usr/local
	cp -dpfR /develop/BUILDROOT/BAY/target_skeleton/opt $(TARGET_DIR)/
	cp -udpR /develop/BUILDROOT/BAY/target_skeleton/etc/* $(TARGET_DIR)/etc
	cp -udpR /develop/BUILDROOT/BAY/target_skeleton/root $(TARGET_DIR)/
	cp -dpfR /develop/BUILDROOT/BAY/target_skeleton/persist $(TARGET_DIR)/
msdpi: uclibc ncurses $(TARGET_DIR)/$(MSDPI_TARGET_BINARY)
msdpi-source: $(DL_DIR)/$(MSDPI_SOURCE)
msdpi-clean:
	$(MAKE) prefix=$(TARGET_DIR)/usr -C $(MSDPI_DIR) uninstall
	-$(MAKE) -C $(MSDPI_DIR) clean
```

Table 5 — Sofia-SIP Buildroot Package Compilation Script

```
SOFIASIP_VERSION:=1.12.10
SOFIASIP_SOURCE:=sofiasip-$(SOFIASIP_VERSION).tar.gz
SOFIASIP_SITE:=http://localhost/BUILDROOT
SOFIASIP_DIR:=$(BUILD_DIR)/sofiasip-$(SOFIASIP_VERSION)
$(DL_DIR)/$(SOFIASIP_SOURCE):
	$(call DOWNLOAD,$(SOFIASIP_SITE),$(SOFIASIP_SOURCE))
$(SOFIASIP_DIR)/.source: $(DL_DIR)/$(SOFIASIP_SOURCE)
	$(ZCAT) $(DL_DIR)/$(SOFIASIP_SOURCE) | tar -C $(BUILD_DIR) $(TAR_OPTIONS) -
	touch $@
$(SOFIASIP_DIR)/.configured: $(SOFIASIP_DIR)/.source
	(cd $(SOFIASIP_DIR); rm -rf config.cache; \
		$(TARGET_CONFIGURE_OPTS) \
		CFLAGS="$(TARGET_CFLAGS)" \
		./configure \
		--target=$(GNU_TARGET_NAME) \
		--host=$(GNU_TARGET_NAME) \
		--build=$(GNU_HOST_NAME) \
		--prefix=/ \
		--includedir=/include \
		--libdir=/lib \
		$(SOFIASIP_CONFIG_SHARED) \
	);
	touch $@
$(SOFIASIP_DIR)/.compiled: $(SOFIASIP_DIR)/.configured
	$(MAKE) -C $(SOFIASIP_DIR)
	touch $@
$(SOFIASIP_DIR)/.installed: $(SOFIASIP_DIR)/.compiled
	$(MAKE) DESTDIR=$(TARGET_DIR) -C $(SOFIASIP_DIR) install-strip
	cp -dpfR $(TARGET_DIR)/include/sofia-resolv $(STAGING_DIR)/usr/include/
	cp -dpfR $(TARGET_DIR)/include/sofia-sip $(STAGING_DIR)/usr/include/
	cp -dpf $(STAGING_DIR)/usr/lib/glib-2.0/include/glibconfig.h
$(STAGING_DIR)/usr/include/glibconfig.h
$(TARGET_DIR)/usr/include/glibconfig.h
	cp -dpf $(SOFIASIP_DIR)/libsofia-sip-ua-glib/.libs/libsofia-sip-ua-glib.so*
$(STAGING_DIR)/usr/lib
	cp -dpf $(SOFIASIP_DIR)/libsofia-sip-ua/.libs/libsofia-sip-ua.a $(STAGING_DIR)/usr/lib/
	cp -dpf $(SOFIASIP_DIR)/libsofia-sip-ua/.libs/libsofia-sip-ua.a $(TARGET_DIR)/usr/lib/
	cp -dpf $(SOFIASIP_DIR)/libsofia-sip-ua-glib/.libs/libsofia-sip-ua-glib.so*
$(STAGING_DIR)/lib
	cp -dpf $(SOFIASIP_DIR)/libsofia-sip-ua/.libs/libsofia-sip-ua.a $(STAGING_DIR)/lib/
	cp -dpf $(SOFIASIP_DIR)/libsofia-sip-ua/.libs/libsofia-sip-ua.a $(TARGET_DIR)/lib/
	touch $@
sofiasip: uclibc $(SOFIASIP_DIR)/.installed
sofiasip-source: $(DL_DIR)/$(SOFIASIP_SOURCE)
sofiasip-clean:
	$(MAKE) prefix=$(TARGET_DIR)/usr -C $(SOFIASIP_DIR) uninstall
	-$(MAKE) -C $(SOFIASIP_DIR) clean
sofiasip-dirclean:
	rm -rf $(SOFIASIP_DIR)
ifeq ($(strip $(BR2_PACKAGE_SOFIASIP)),y)
TARGETS+=sofiasip
endif
```

Table 6 — MSDPI Buildroot Package Config.in Menu Script

```
config BR2_PACKAGE_MSDPI
        bool “msdpi”
        help
          Multi-Service Domain Protecting Interface.
          http://localhost/BUILDROOT
```

Table 7 — Sofia-SIP Buildroot Package Menu Script

```
config BR2_PACKAGE_SOFIASIP
        bool “sofiasip”
        help
          Sofia SIP library.
          http://sofia-sip.sourceforge.net/download.html
```

14.5.4 Shortcomings to the Bay Microsystems Implementation of Buildroot

The Buildroot is always evolving to increase the feature sets it includes in its distribution so it supports more applications and kernel updates and enhancements. Particularly, it provides new updates that support hardware-specific additions as more and more product vendors incorporate Buildroot as their product line operating system. This requires hardware manufacturers to actively provide updates to any current Buildroot updates.

Further, MSDPI and Sofia-SIP exploit libraries that are incorporated into the latest Buildroot releases. Unfortunately, the Bay Microsystems implementation of Buildroot corresponds to a release of Buildroot prior to 2009, which does not support these libraries.

More important, Buildroot is updated to support the most current and stable release of Linux. Bay's distribution is currently (as of this writing) built against the Linux 2.6.19 release. MSDPI is built against Buildroot release 2011 which supports a newer Linux release. Obviously the benefits to a newer Buildroot include security enhancements as well as feature set enhancements such as utility libraries like “readline” (only one of many MSDPI exploited libraries). However, in attempts to fully incorporate MSDPI within the Bay product, the latest Buildroot Linux distribution was replaced with a Bay-supported Linux release. Although this was successfully accomplished, what could not be completed was the inclusion of Bay's specific hardware drivers source code for the ABEx/NP10. This source code was not made available.

This has not prevented MSDPI from being cross-compiled for the ABEx/NP10 devices while still using the 2011 Buildroot release. But it does prevent MSDPI from being completely integrated into the ABEx/NP10 boot image, requiring instead that MSDPI be manually loaded into the device. Besides increasing deployment time, this also has the result that MSDPI is not persistent at ABEx/NP10 reload.

14.6 NRL MSDPI DISN Policy Proxy Prototype

Figure 31 illustrates how the MSDPI has been developed to function as a policy proxy device for a router used by the DoD within its Defense Information Systems Network (DISN) backbone network. Here the MSDPI interfaces to a Juniper 10i via Juniper's JUNOScript interface. MSDPI simply translates its exchanged policy PIDF data into JUNOScript messages which are transmitted to a Juniper router via its management interface. Since JUNOScript is also XML formatted messages, the translation process is seamless.

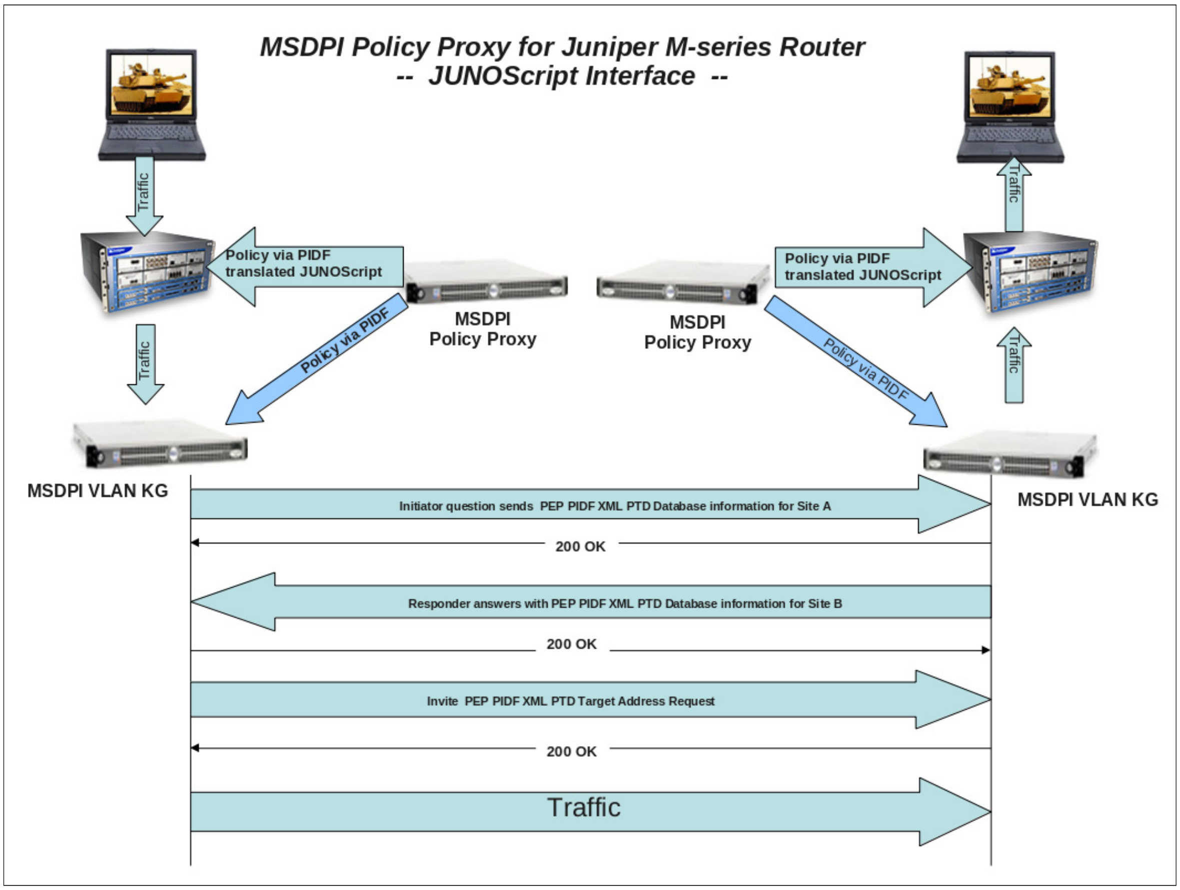


Fig. 31 — DoD DISN policy proxy prototype

Table 8 consists of panels containing an extraction from the MSDPI code which is responsible for communicating with a Juniper router through its management Ethernet interface. Panel 1 (reading left to right) contains the series of canned XML tagged strings used to login into the router. Panel 2 shows an “rpc” XML tagged load-configuration command which sets the router’s IP address. Just like the MSDPI XML protocol, JUNOScript XML files contain tagged formatted commands. The following is a typical JUNOScript tagged command:

```
<junoscript>
   <rpc [attributes]>
     <!-tag elements comment -  ->
            <interface-state>enabled</interface-state>
            <input-bytes>25378</input-bytes>
    </rpc>
</junoscript>
```

Table 8 — MSDPI Juniper Policy Proxy Program Code

```
char *loginLine1 =
“<?xml version=\”1.0\” encoding=\”us-ascii\”?>\0”;
char *loginLine2 =
“<junoscript version=\”1.0\” hostname=\”MSDPI\”
release=\”8.1R1\”>\0”;
char *loginPass1 =
“<rpc><request-login><username>\0”;
char *loginPass2 =
“</username><challenge-response>\0”;
char *loginPass3 =
“</challenge-response></request-login></rpc>\0”;
```

```
char *setConfigLine1 =
“<rpc><load-configuration action=\”merge\”>
<configuration>\<interfaces>
<interface>
<name>ge-1/0/0</name>
<unit>\<name>0</name><family><inet><address>\<name>10.133.13
3.1/24</name></address></inet></family></unit>
</interface></interfaces></configuration>
</load-configuration>
</rpc>\0”;
```

```
pthread_join(clientJuniperLoginHandlerThreadID, NULL);
while(xmtCmdlinePtrs[sentRow]){
 send(juniperConnectionSocket,xmtCmdlinePtrs[sentRow],
    strlen(xmtCmdlinePtrs[sentRow]), 0);
 while(1){
   memset(&currentRcvBuffer[0], 0, currentRcvBufferLength);
   if((recvCharacterCount=recv(juniperConnectionSocket,
     &currentRcvBuffer[0],currentRcvBufferLength, 0)) < 0){
     pthread_exit(NULL);}
   else{
     receivePtrs = receiveBufferPtrs;
     while(receivePtrs){
       previousReceivePtrs = receivePtrs;
       receivePtrs = receivePtrs->next;}
     receivePtrs = previousReceivePtrs;
     if((newReceivePtrs=malloc(sizeof(struct
receiveBuffer)))==NULL){
       printf(“Failed to allocate a receive buffer\n”);
       pthread_exit(NULL);}
```

```
if((newReceivePtrs->buffer =
       (char *)malloc(recvCharacterCount)) == NULL){
      printf(“Failed to allocate a receive buffer\n”);
      free(newReceivePtrs);
      pthread_exit(NULL);}
    receivePtrs->next = newReceivePtrs;
    newReceivePtrs->associatedReceiveSocket = 0;
    newReceivePtrs->next = NULL;
    strncpy(newReceivePtrs->buffer,&currentRcvBuffer[0],
          recvCharacterCount);
    newReceivePtrs->bufferLength = recvCharacterCount;
    newReceivePtrs->associatedReceiveSocket =
juniperConnectionSocket;
    if(strstr(newReceivePtrs->buffer,”</rpc-reply>“) != NULL){
      break;}}}
  if((strstr(newReceivePtrs->buffer,”<load-configuration-
results>“)!=NULL)&&
    (strstr(newReceivePtrs->buffer,”<load-success/>“) != NULL)){
   commitJuniperCommand = 0;}
  else{commitJuniperCommand = -1;}
  sentRow++;}
  printf(“\n\n%s\n”,(char *)&rcvReply[0]);
```

Table 8 (cont.) — MSDPI Juniper Policy Proxy Program Code

```
if(commitJuniperCommand==0){
  char *commitCommand = “<rpc><commit-
configuration><check/></commit-configuration></rpc>“;
  send(juniperConnectionSocket,commitCommand,
strlen(commitCommand),0);
  while(1){
    memset(&currentRcvBuffer[0], 0, currentRcvBufferLength);
    if((recvCharacterCount=recv(juniperConnectionSocket,
       &currentRcvBuffer[0],currentRcvBufferLength, 0))<0){
      pthread_exit(NULL);}
    else{
      receivePtrs=receiveBufferPtrs;
      while(receivePtrs){
        previousReceivePtrs = receivePtrs;
        receivePtrs = receivePtrs->next;}
      receivePtrs = previousReceivePtrs;
      if((newReceivePtrs=malloc(sizeof(struct
receiveBuffer)))==NULL){
        printf(“Failed to allocate a receive buffer\n”);
        pthread_exit(NULL);}
     if((newReceivePtrs->buffer=
            malloc(recvCharacterCount))== NULL){
        printf(“Failed to allocate a receive buffer\n”);
        free(newReceivePtrs);
        pthread_exit(NULL);}
      receivePtrs->next = newReceivePtrs;
      newReceivePtrs->associatedReceiveSocket = 0;
```

```
       newReceivePtrs->next = NULL;
      strncpy(newReceivePtrs->buffer,&currentRcvBuffer[0],
              recvCharacterCount);
      newReceivePtrs->bufferLength=recvCharacterCount;
      newReceivePtrs-
>associatedReceiveSocket=juniperConnectionSocket;
      if(strstr(newReceivePtrs->buffer,”</rpc-reply>“) != NULL){
        if(strstr((char *)newReceivePtrs->buffer,”<commit-check-
success/>“)!=NULL){
          printf(“Commit check successful with: %s\n”,
                 newReceivePtrs->buffer);}
        break;}
      else if(strstr((char *)newReceivePtrs-
>buffer,”<status>fail</status>“)!=NULL){
        printf(“Dialogue failure.\n”);
        pthread_exit(NULL);}
      else if(strstr((char *)newReceivePtrs->buffer,”</junoscript”) !=
NULL){
        printf(“Dialogue terminated with: %s\n”,
               newReceivePtrs->buffer);
        pthread_exit(NULL);}
```

```
    else if(strstr(newReceivePtrs->buffer,”<commit-check-success/>“)
!= NULL){
      printf(“Commit check successful with: %s\n”,newReceivePtrs-
>buffer);
        break;}}}
char *commitIt = “<rpc><commit-configuration/></rpc>“;
send(juniperConnectionSocket,commitIt, strlen(commitIt), 0);
while(1){
  memset(&currentRcvBuffer[0], 0, currentRcvBufferLength);
  if((recvCharacterCount=recv(juniperConnectionSocket,
      &currentRcvBuffer[0],currentRcvBufferLength,0))<0){
    pthread_exit(NULL);}
  else{
    receivePtrs = receiveBufferPtrs;
    while(receivePtrs){
      previousReceivePtrs = receivePtrs;
      receivePtrs = receivePtrs->next;}
    receivePtrs = previousReceivePtrs;
    if((newReceivePtrs=malloc(sizeof(struct receiveBuffer)))==
       NULL){
      printf(“Failed to allocate a receive buffer\n”);
      pthread_exit(NULL);}
```

```
  if((newReceivePtrs->buffer =
       malloc(recvCharacterCount)) == NULL){
      printf(“Failed to allocate a receive buffer\n”);
      free(newReceivePtrs);
      pthread_exit(NULL);}
    receivePtrs->next = newReceivePtrs;
    newReceivePtrs->associatedReceiveSocket = 0;
    newReceivePtrs->next = NULL;
    strncpy(newReceivePtrs->buffer,
          (char *)&currentRcvBuffer[0],
          recvCharacterCount);
    newReceivePtrs->bufferLength = recvCharacterCount;
    newReceivePtrs->associatedReceiveSocket =
juniperConnectionSocket;
    if(strstr(newReceivePtrs->buffer,”</rpc-reply>“)!= NULL){
      if(strstr(newReceivePtrs->buffer,”<commit-success/>“) !=
NULL){
        printf(“Commit successful with: %s\n”,newReceivePtrs-
>buffer);}
      break;}
    else if(strstr((char *)newReceivePtrs-
>buffer,”<status>fail</status>“)!=NULL){
      printf(“Dialogue failure.\n”);
      pthread_exit(NULL);}
```

```
 else if(strstr((char *)newReceivePtrs-
>buffer,”</junoscript”)!=NULL){
   printf(“Dialogue terminated with: %s\n”,newReceivePtrs->buffer);
   pthread_exit(NULL);}
 else if(strstr((char *)newReceivePtrs->buffer,”<commit-
success/>“)!=NULL){
   printf(“Commit successful with: %s\n”,newReceivePtrs->buffer);
 }}}}
pthread_exit(NULL);
}
```

15 MSDPI AS A TEST SUITE

The MSDPI has proven to work well as a network test suite. Figure 32 demonstrates one of the successful test runs using the MSDPI as a Test Master (server) and Test Client system for conducting the InfiniBand test "ib_send_bw" (or sometimes referenced as "send_bw"). The MSDPI Test Master, using the MSDPI protocol, transmits a test configuration PIDF to each MSDPI Test Client. The clients process the configurations becoming either a listener or sender for the test. In this case the test is the ib_send_bw. The Test Client's MSDPI reads the PIDF files and spawns off either a listener ib_send_bw or sender ib_send_bw. Each of these spawned subsystems functions exactly as the standalone version of ib_send_bw[15] except the data is reported in a format suited for translation to a typical spreadsheet application. Additionally, the MSDPI "perftest" utilities[16] have been enhanced to analyze and report on IB call setup performance. This feature is not available with the standalone version of the perftest suite, found only in the MSDPI subsystem version of the utilities. Further, the MSDPI utilities have the ability to repeat each test, to include call setup as well as the iteration sequences. This can been seen in Fig. 33, "Run Count," which shows an executed test count of 1,146,653 runs with call setups and 1,146,653,000 test iterations, that is 1,000 iterations for each call setup.

The perftest tools consist of the following standalone utilities: send_bw, send_lat, read_bw, read_lat, write_bw, write_lat, rdma_bw, and rdma_lat. Two of these utilities had particular interest for the initial testing to be performed: send_bw and write_bw. Both these utilities send a stream of data from IB queue-pairs (QPs). The main difference between these two is how the QP sending process is performed. send_bw establishes a connection between IB hosts, then transmits a single QP. write_bw transmits a group of QPs per connection. Since send_bw would exercise both MSDPI subsystem processes and network bandwidth limitations, to include system interfaces and IB data transfer setup, it was selected as the first utility to be integrated into MSDPI.

Although the perftest tools function without problems as standalone utilities, there were several changes required before the tool suite could be integrated into the MSDPI system. It should be noted, all the utilities are basically structured the same, that is, programming changes to one can be incorporated into the others without too many changes. However, the changes for the first selected utility, in this case send_bw, were substantial, resulting in unanticipated code development and testing. The most troubling correction was that almost all memory allocation APIs required correction to prevent early termination of the send_bw process when integrated into MSDPI. The next change required rewriting send_bw so it was not a standalone utility but a subsystem to MSDPI. This meant modifications to the system main function, essentially removing the main function and reworking all the command line inputs as subsystem parameters. Additionally, APIs were developed between the MSDPI core system and the newly created send_bw subsystem. This included SIP message processing and send_bw test data database processing. Next, new timer routines were created to allow for synchronized testing between multiple MSDPI test systems. A new timer feature allowed the tester to queue up several systems to begin testing at a specific time. Another change was the reporting process. Fig. 34 illustrates the new reporting format which now includes μsec time data.

Table 9 lists the listener command directives and Table 10 lists the sender command directives when bundled into a test's SIP message PIDF document. Most of the tagged[17] directives are self explanatory. A

---

[15] See the OFED for details on how the InfiniBand "perftest" utilities operate.
[16] The "perftest" utilities typically include ib_send_bw, ib_send_lat, ib_read_bw, ib_read_lat, ib_write_bw, ib_write_lat, ib_rdma_bw and ib_rdma_lat.
[17] The reader should become familiar with the XML format to understand how the MSDPI exploits XML tags.

few should be noted such as the <current_time> and <start_time> tags. The <current_time>[18] tag is used by the MSDPI server to synchronize all client time to NTP time. The <start_time> informs the client to begin the test processing at the designated time. <start_time> places the client in a wait-until-time-expires loop. The <command> tag denotes the perftest to run and the <arguments> tag lists the command options as found in the typical OFED perftest standalone application. As with the OFED version of the perftest tools, test clients must be TCP Port paired. To clarify which listener a sender is peering with, an additional command-line option "-h" was added. This option simply designates the listener's IP address.

The last figure, Fig. 35, contains an example of an MSDPI ib_send_bw test and some of the reporting data. This report details how the MSDPI ib_send_bw tool was used to report on the performance of a QSFP-to-CX4 cable configuration. The test also demonstrated some problems with the OFED release which were repaired in the MSDPI version. Fig. 34 is an example of typical ib_send_bw test data collected and ready for formatting into a final test report.

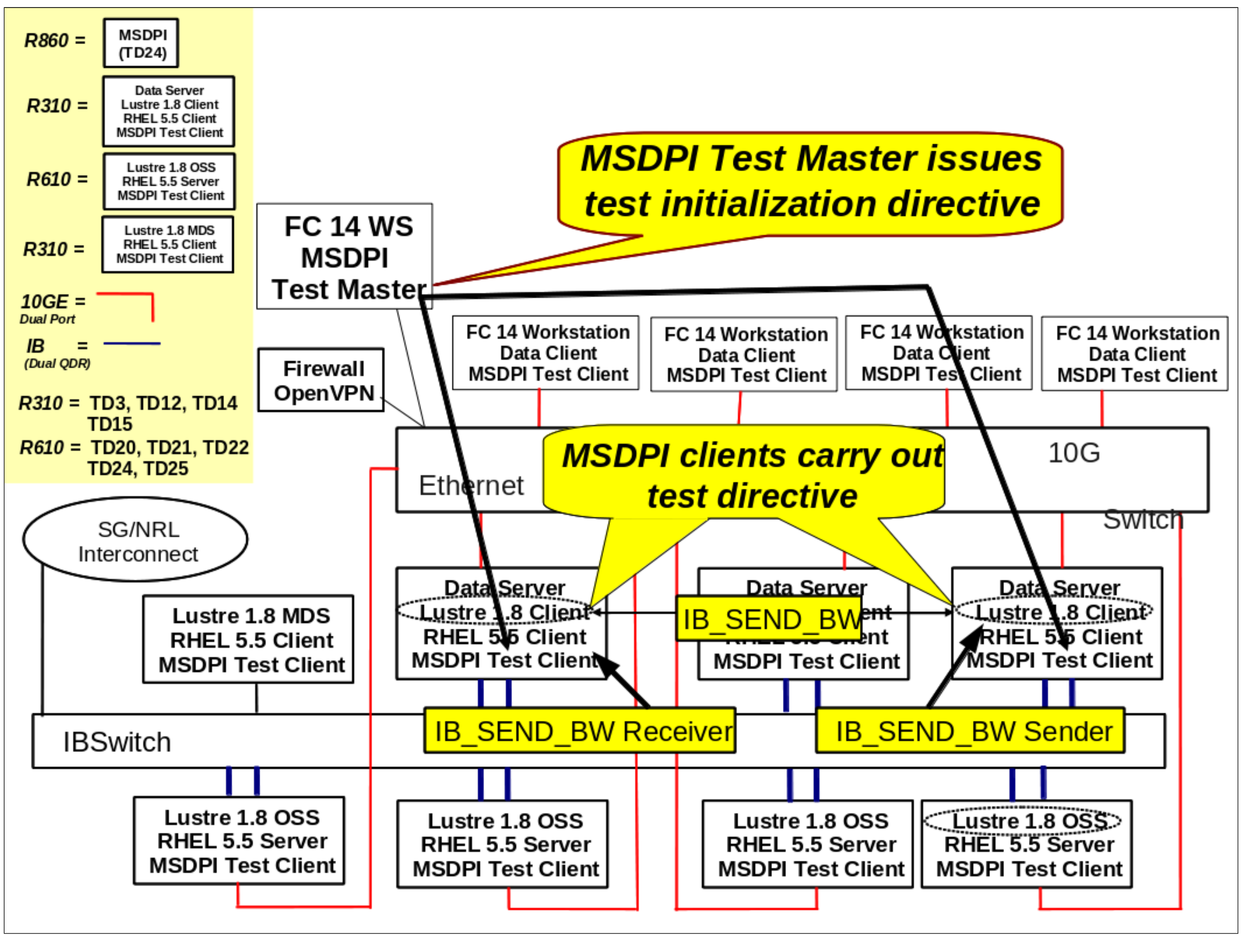


Fig. 32 — MSDPI Test Master/Client configuration of IB tests

[18] The <current_time> feature has not been fully implemented in the first release of MSDPI; therefore, it is assumed all MSDPI clients are NTP clock synchronized.

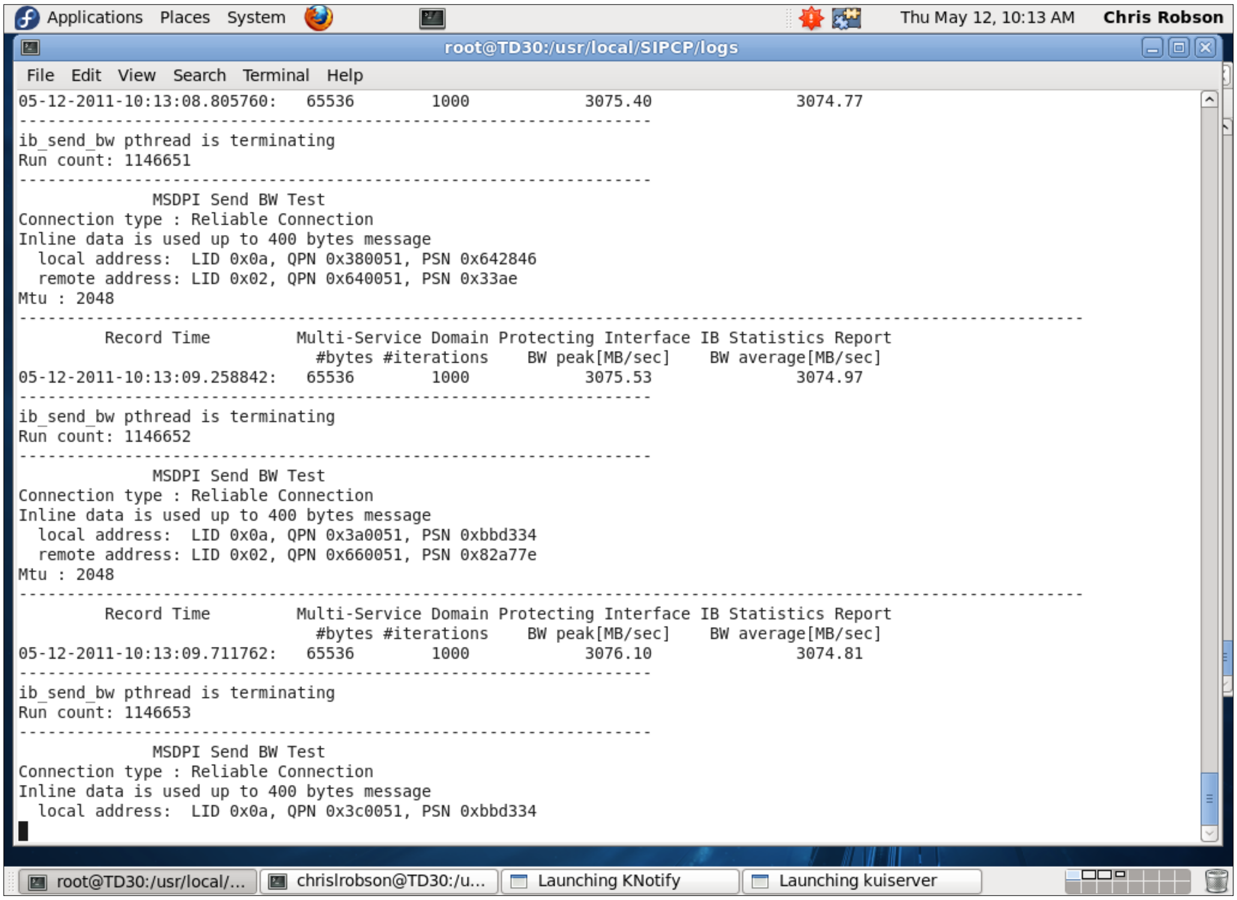
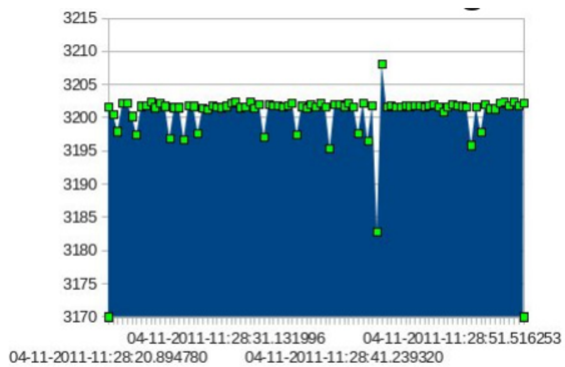


Fig. 33 — Captured MSDPI IB test results

TIME TEST RAN	No. BYTES SENT	ITER	PEAK BW	AVG BW
04-11-2011-11:28:20.894780:	65536	1000	3201.63	3200.00
04-11-2011-11:28:21.208097:	65536	1000	3200.46	2147.42
04-11-2011-11:28:21.672812:	65536	1000	3197.93	3196.36
04-11-2011-11:28:21.979729:	65536	1000	3202.12	3200.40
04-11-2011-11:28:22.556627:	65536	1000	3202.12	3200.45
04-11-2011-11:28:22.997049:	65536	1000	3200.21	3194.69
04-11-2011-11:28:23.454400:	65536	1000	3197.38	3194.31
04-11-2011-11:28:23.764796:	65536	1000	3201.69	3200.04
04-11-2011-11:29:19.560067:	65536	1000	3201.63	3199.93
04-11-2011-11:29:19.996219:	65536	1000	3201.75	3200.10
04-11-2011-11:29:20.458303:	65536	1000	3201.81	3200.14
04-11-2011-11:29:20.962138:	65536	1000	3196.27	3194.63

Fig. 34 — Captured output of MSDPI IB test results



Demonstrated benefits from this test:
•µsec collection of data points
•Improved IB test performance
•Controlled execution time at real time
•Remote control of testing systems
•Efficient, simplified test scripting
•True performance (call setup)

Demonstrated problems with this test:
•Open source IB tool “ib_send_bw”
- Not thread safe (rewriting)
- Memory leaks (fixed)
- 7115 loop count failure (send complete memory leak?)

Fig. 35 — Captured MSDPI IB QSFP-to-CX4 cable test results

Table 9 — MSDPI PIDF - ib_send_bw Listener Configuration File


```
<?xml version=“1.0” encoding=“ISO-8859-1”?>
<system_commands>
<system>
<system_address>10.2.1.202</system_address>
<system_prefixlength>24</system_prefixlength>
<system_ttl>120</system_ttl>
<application>
<current_time>05-05-2011 06:15:00</current_time>
<current_time_usec>0000</current_time_usec>
<start_time>05-05-2011 06:15:00</start_time>
<start_time_usec>0000</start_time_usec>
<command>ib_send_bw</command>
<arguments>-p 19024 -b -n 1000 -m 2048 -I 400 -s 65536</arguments>
<mode>server</mode>
</application>
<system_command_poc>
<name>NRL</name>
<phone>111-111-1111</phone>
</system_command_poc>
</system>
<checksum>SHA-123456789</checksum>
</system_commands>
```

Table 10 — MSDPI PIDF - ib_send_bw Sender Configuration File


```
<?xml version="1.0" encoding="ISO-8859-1"?>
<system_commands>
<system>
<system_address>10.2.1.202</system_address>
<system_prefixlength>24</system_prefixlength>
<system_ttl>120</system_ttl>
<application>
<current_time>04-11-2011 05:00:00</current_time>
<current_time_usec>0000</current_time_usec>
<start_time>04-11-2011 11:04:00</start_time>
<start_time_usec>0000</start_time_usec>
<command>ib_send_bw</command>
<arguments>-p 19300 -h 10.2.1.218</arguments>
<mode>client</mode>
</application>
<system_command_poc>
<name>NRL</name>
<phone>111-111-1111</phone>
</system_command_poc>
</system>
<checksum>SHA-123456789</checksum>
</system_commands>
```


16 MSDPI L2TPV3 TEST FOR LARGE DATA JCTD

The following section outlines a modification to MSDPI to support a test for the Large Data Joint Capability Technology Demonstration (JCTD) office. The purpose of the test was to validate the proper operation of L2TPv3 within the Bay Microsystems ABEx network device. This MSDPI modification is an example of MSDPI using the "system()" function to spawn a program that is not fully integrated into the MSDPI core process. When MSDPI calls a program via the system() function, it is run as a separate shell command.

16.1 L2TPv3

Layer 2 Tunneling Protocol Version 3 is for encapsulation of multiple protocols within a Layer 2 communications packet to traverse over IP networks. L2TPv3 provides a pseudo-wire service which scales to fit within carrier requirements.

L2TPv3 can be regarded as being to MPLS what IP is to ATM: a simplified version of the same concept, with much of the goodness achieved with a fraction of the effort, at the cost of losing some technical features considered less important in the market. In the case of L2TPv3, the features lost are teletraffic engineering features considered important in MPLS. The protocol overhead of L2TPv3 is also significantly bigger than for MPLS. However, there is no reason why these features cannot be re-engineered in or on top of L2TPv3 in later products.

16.2 Raw Collected Data

The following sections provide detailed Netperf[19] data collection from the MSDPI L2TPv3 test. In all charts of performance results, the horizontal axis is test number and the vertical axis is bandwidth.

16.2.1 Device Comparison Study

Table 11 defines six configurations that were to be tested; five of the six were tested. Figure 36 and Fig. 37 show summary data from the five scenarios. The sections ahead provide more detailed data for each test.

Table 11 — Test Configurations for the MSDPI L2TPv3 Test

B2BHost	Two hosts directly connected
HKGH	Two hosts connected through two network encryption devices
HABExH	Two hosts connected through two Bay Microsystems ABEx network devices
HCKGCH	Two hosts connected through two Cisco ASR1004 routers through two encryption devices
HCABExCH	Two hosts connected through two Cisco ASR1004 routers and two Bay Microsystems ABEx network devices
HCABExKGABExCH	Two hosts connected through two Cisco ASR1004 routers through two Bay Microsystems ABEx network devices and two encryption devices

[19] (Extracted from man) Netperf is a benchmark that can be used to measure various aspects of networking performance. Currently, its focus is on bulk data transfer and request/response performance using either TCP or UDP, and the Berkeley Sockets interface. In addition, tests for DLPI, and Unix Domain Sockets, tests for IPv6 may be conditionally compiled-in.

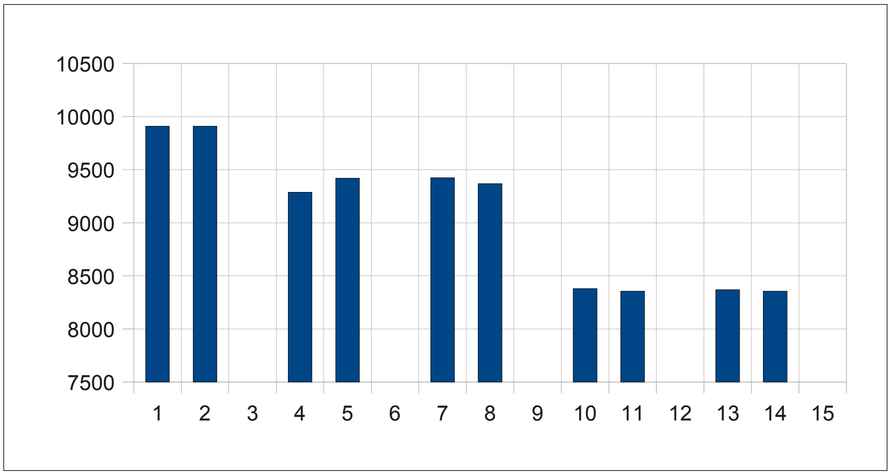


Fig. 36 — Test results from the five configurations

Test	Recv Socket Size bytes	Xmt Socket Size bytes	Xmt Msg Size bytes	Elpse Time secs.	Thruput 10^6b/s	Xmt CPU Util	Recv CPU Util	Xmt Local µs/KB	Recv remote µs/KB	TCP_STREAM (transmit) TCP_MAERTS (received)
1	87380	65536	65536	60.01	9909.52	3.4	4.07	0.225	0.269	B2BHost xmit
2	87380	65536	65536	60.01	9910.27	4.98	3.78	0.329	0.25	B2BHost recv
4	87380	65536	65536	60	9287.49	17.02	44.01	0.601	0.776	HKGH xmit
5	87380	65536	65536	60.01	9419.26	19.14	35.71	0.666	0.621	HKGH recv
7	87380	65536	65536	60.01	9423.16	2.77	3.81	0.192	0.265	HABExH xmit
8	87380	65536	65536	60.01	9369.32	4.8	3.67	0.336	0.257	HABExH recv
10	87380	65536	65536	60.02	8379.84	16.1	44.5	0.63	0.87	HCKGCH xmit
11	87380	65536	65536	60.02	8355.83	20.39	31.56	0.8	0.619	HCKGCH recv
13	87380	65536	65536	60.01	8368.92	15.7	44.79	0.615	0.877	HCH xmit
14	87380	65536	65536	60.02	8355.57	20.23	32.12	0.793	0.63	HCH recv

Fig. 37 — Summary of L2TPv3 test results

16.2.2 Host-to-Host Study

Table 12 and Fig. 38 present host-to-host performance data (B2BHost). This circuit has no network devices or encryption devices. Therefore no packet data manipulation was performed.

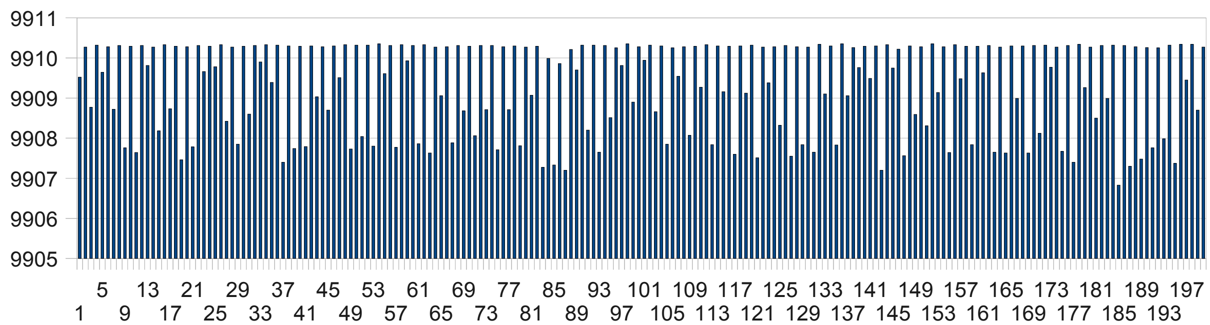


Fig. 38 — Host-to-host performance results (B2BHost)

Table 12 — Host-to-Host Test Data

Test	Recv Socket Size bytes	Xmt Socket Size bytes	Xmt Msg Size bytes	Elpse Time secs.	Thruput 10^6b/s	Xmt CPU Util	Recv CPU Util	Xmt Local µs/KB	Recv remote µs/KB	TCP_STREAM (transmit) TCP_MAERTS (received)
1	87380	65536	65536	60.01	9909.52	3.40	4.07	0.225	0.269	Transmit
2	87380	65536	65536	60.01	9910.27	4.98	3.78	0.329	0.250	Received
3	87380	65536	65536	60.01	9908.77	3.04	3.88	0.201	0.257	Transmit
4	87380	65536	65536	60.01	9910.32	4.94	3.64	0.327	0.241	Received
5	87380	65536	65536	60.01	9909.64	3.31	3.87	0.219	0.256	Transmit
6	87380	65536	65536	60.01	9910.28	5.51	3.75	0.364	0.248	Received
7	87380	65536	65536	60.01	9908.72	3.29	4.05	0.218	0.268	Transmit
8	87380	65536	65536	60.01	9910.31	5.17	3.67	0.342	0.242	Received
9	87380	65536	65536	60.01	9907.76	3.22	3.85	0.213	0.254	Transmit
10	87380	65536	65536	60.01	9910.29	4.98	3.63	0.329	0.240	Received
11	87380	65536	65536	60.01	9907.64	3.40	3.95	0.225	0.261	Transmit
12	87380	65536	65536	60.01	9910.31	5.05	3.77	0.334	0.249	Received
13	87380	65536	65536	60.01	9909.81	3.32	3.98	0.219	0.263	Transmit
14	87380	65536	65536	60.02	9910.27	5.13	3.63	0.339	0.240	Received

16.2.3 Host–KG–KG–Host Study

Table 13 and Fig. 39 present host-to-host through back-to-back KGs performance data (HKGH). This circuit has back-to-back Level 3 KG-245X devices between each host. This study was limited to Dell 860 hosts; however, the host's PCIe Gen2 slot was used with Myricom 10GE Fiber XFPs.

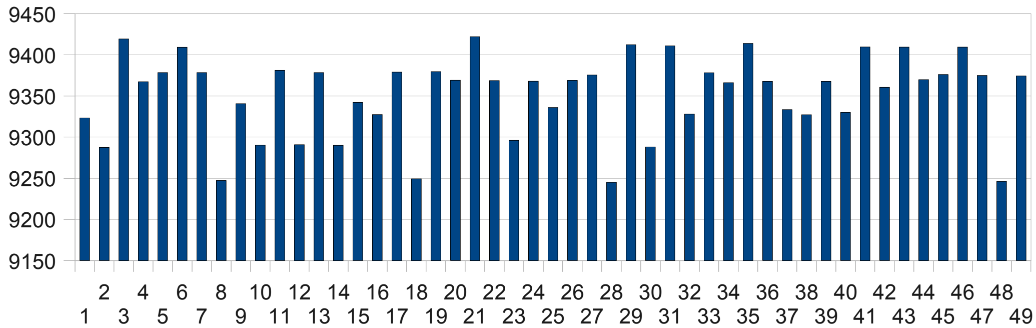


Fig. 39 — Host–KG–KG–Host performance results (HKGH)

Table 13 — Host–KG–KG–Host Test Data

Test	Recv Socket Size bytes	Xmt Socket Size bytes	Xmt Msg Size bytes	Elpse Time secs.	Thruput 10^6b/s	Xmt CPU Util	Recv CPU Util	Xmt Local µs/KB	Recv remote µs/KB	TCP_STREAM (transmit) TCP_MAERTS (received)
1	87380	65536	65536	60.00	9323.19	16.88	44.07	0.593	0.775	Transmit
2	87380	65536	65536	60.00	9287.49	17.02	44.01	0.601	0.776	Transmit
3	87380	65536	65536	60.01	9419.26	19.14	35.71	0.666	0.621	Received
4	87380	65536	65536	60.01	9367.24	17.05	44.52	0.596	0.779	Transmit
5	87380	65536	65536	60.01	9378.41	19.12	35.71	0.668	0.624	Received
6	87380	65536	65536	60.00	9409.22	17.01	44.70	0.592	0.778	Transmit
7	87380	65536	65536	60.01	9378.50	19.18	35.70	0.670	0.624	Received
8	87380	65536	65536	60.00	9247.16	16.94	44.36	0.600	0.786	Transmit
9	87380	65536	65536	60.01	9340.68	19.29	35.56	0.677	0.624	Received
10	87380	65536	65536	60.01	9290.24	16.89	44.60	0.596	0.787	Transmit
11	87380	65536	65536	60.01	9381.07	19.25	35.91	0.672	0.627	Received
12	87380	65536	65536	60.00	9290.76	16.91	44.33	0.596	0.782	Transmit
13	87380	65536	65536	60.01	9378.46	19.22	35.78	0.671	0.625	Received
14	87380	65536	65536	60.01	9289.96	16.91	44.42	0.596	0.783	Transmit

16.2.4 Bay Microsystems Baseline Study

Table 14 and Fig. 40 present raw performance data collected with the Bay Microsystems ABEx 5010 inserted into the test bed circuit: host–ABEx–ABEx–host (HABExH).

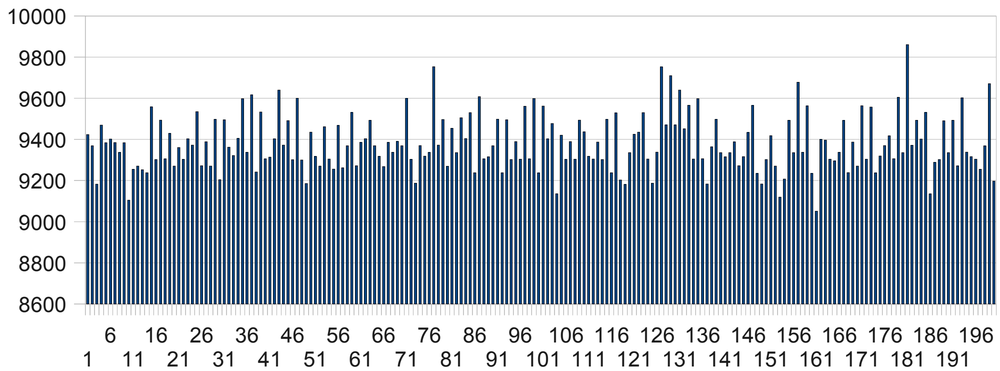


Fig. 40 — Performance results for two hosts connected through two Bay Microsystems ABEx network devices (HABExH)

Table 14 — Host–ABEx–ABEx–Host Test Data

Test	Recv Socket Size bytes	Xmt Socket Size bytes	Xmt Msg Size bytes	Elpse Time secs.	Thruput 10^6b/s	Xmt CPU Util	Recv CPU Util	Xmt Local µs/KB	Recv remote µs/KB	TCP_STREAM (transmit) TCP_MAERTS (received)
1	87380	65536	65536	60.01	9423.16	2.77	3.81	0.192	0.265	Transmit
2	87380	65536	65536	60.01	9369.32	4.80	3.67	0.336	0.257	Received
3	87380	65536	65536	60.01	9182.00	2.90	3.86	0.207	0.276	Transmit
4	87380	65536	65536	60.01	9469.60	5.11	3.77	0.354	0.261	Received
5	87380	65536	65536	60.01	9383.65	2.59	3.77	0.181	0.263	Transmit
6	87380	65536	65536	60.01	9401.74	5.37	3.71	0.374	0.259	Received
7	87380	65536	65536	60.01	9384.96	2.80	3.75	0.196	0.262	Transmit
8	87380	65536	65536	60.01	9336.22	5.42	3.88	0.380	0.272	Received
9	87380	65536	65536	60.01	9384.22	2.69	3.77	0.188	0.263	Transmit
10	87380	65536	65536	60.01	9104.79	5.00	3.69	0.360	0.266	Received
11	87380	65536	65536	60.01	9254.84	2.55	3.67	0.180	0.260	Transmit
12	87380	65536	65536	60.01	9270.70	5.21	3.71	0.368	0.262	Received
13	87380	65536	65536	60.01	9252.64	2.79	3.87	0.197	0.274	Transmit
14	87380	65536	65536	60.01	9237.59	4.97	3.68	0.352	0.261	Received

16.2.5 Cisco Baseline Study

Table 15 and Fig. 41 present raw performance data collected with back-to-back Cisco ASR1004 routers inserted into the test bed circuit: host–ASR–KG–KG–ASR–host (HCKGCH).

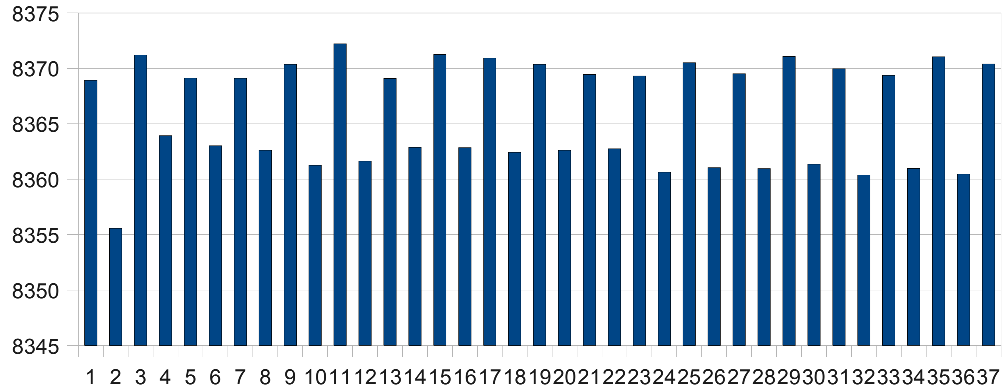


Fig. 41 — Performance data for two hosts connected through two CISCO ASR1004 routers through two encryption devices (HCKGCH)

Table 15 — Host–ASR–KG–KG–ASR–Host Test Data

Test	Recv Socket Size bytes	Xmt Socket Size bytes	Xmt Msg Size bytes	Elpse Time secs.	Thruput 10^6b/s	Xmt CPU Util	Recv CPU Util	Xmt Local µs/KB	Recv remote µs/KB	TCP_STREAM (transmit) TCP_MAERTS (received)
1	87380	65536	65536	60.01	8368.92	15.70	44.79	0.615	0.877	Transmit
2	87380	65536	65536	60.02	8355.57	20.23	32.12	0.793	0.630	Received
3	87380	65536	65536	60.01	8371.20	15.69	44.82	0.614	0.877	Transmit
4	87380	65536	65536	60.02	8363.93	20.37	31.70	0.798	0.621	Received
5	87380	65536	65536	60.02	8369.13	15.75	44.82	0.617	0.877	Transmit
6	87380	65536	65536	60.02	8363.02	20.50	31.74	0.803	0.622	Received
7	87380	65536	65536	60.02	8369.12	15.75	44.74	0.617	0.876	Transmit
8	87380	65536	65536	60.02	8362.62	20.50	31.73	0.803	0.622	Received
9	87380	65536	65536	60.01	8370.36	15.72	44.87	0.615	0.878	Transmit
10	87380	65536	65536	60.02	8361.25	20.49	31.70	0.803	0.621	Received
11	87380	65536	65536	60.01	8372.21	15.82	44.93	0.619	0.879	Transmit
12	87380	65536	65536	60.02	8361.63	20.56	31.78	0.806	0.623	Received
13	87380	65536	65536	60.01	8369.08	15.70	44.82	0.615	0.878	Transmit
14	87380	65536	65536	60.02	8362.87	20.66	31.81	0.810	0.623	Received

16.2.6 Host, L2TPv3 VPN Device and Router Study

This test (HCABExCH), in which two hosts were to be connected through Cisco ASR1004 routers and two Bay Microsystems ABEx network devices, was not performed. Therefore, there was no resolution for the previous test, the Cisco baseline test.

16.2.7 Host, L2TPv3 VPN Device, Router with Network Encryption Device Study

Table 16 and Fig. 42 present raw performance data collected from two Dell 860 hosts connected through two Cisco ASR1004 routers, two Bay Microsystems ABEx network devices, and two Level3 Red Eagle KG-245X devices (HCABExKGABExCH).

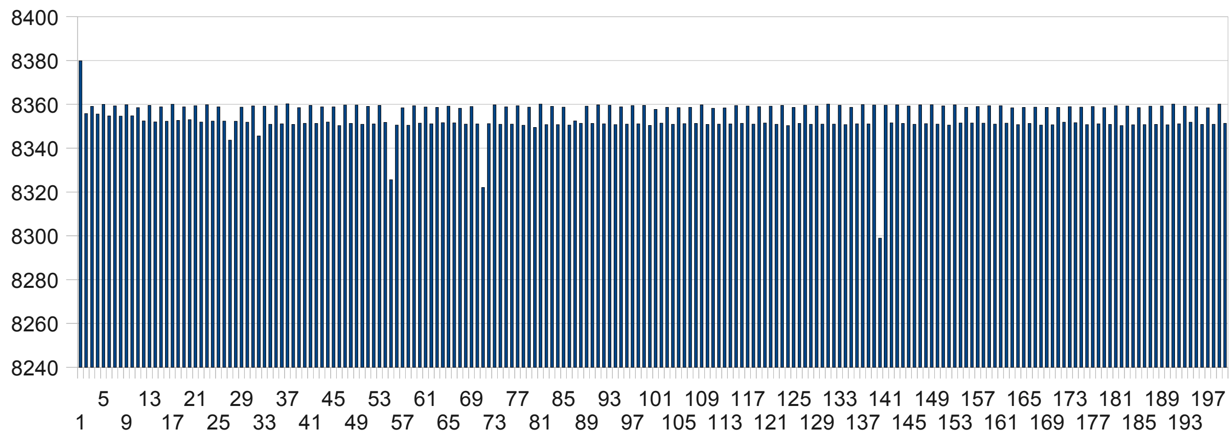


Fig. 42 — Performance data collected from two Dell 860 hosts connected through two CISCO ASR1004 routers, two Bay Microsystems ABEx network devices, and two Level3 Red Eagle KG-245X encryption devices (HCABExKGABExCH)

Table 16 — Host–ASR–ABEx–KG–KG–ABEx–ASR–Host Test Data

Test	Recv Socket Size bytes	Xmt Socket Size bytes	Xmt Msg Size bytes	Elpse Time secs.	Thruput 10^6b/s	Xmt CPU Util	Recv CPU Util	Xmt Local µs/KB	Recv remote µs/KB	TCP_STREAM (transmit) TCP_MAERTS (received)
1	87380	65536	65536	60.02	8379.84	16.10	44.50	0.630	0.870	Transmit
2	87380	65536	65536	60.02	8355.83	20.39	31.56	0.800	0.619	Received
3	87380	65536	65536	60.02	8359.12	15.55	44.42	0.610	0.871	Transmit
4	87380	65536	65536	60.02	8355.62	20.45	31.57	0.802	0.619	Received
5	87380	65536	65536	60.01	8359.99	15.52	44.20	0.608	0.866	Transmit
6	87380	65536	65536	60.02	8354.70	20.40	31.51	0.800	0.618	Received
7	87380	65536	65536	60.01	8359.32	15.52	44.37	0.609	0.870	Transmit
8	87380	65536	65536	60.02	8354.61	20.50	31.64	0.804	0.621	Received
9	87380	65536	65536	60.01	8359.79	15.54	44.55	0.609	0.873	Transmit
10	87380	65536	65536	60.02	8354.76	20.49	31.61	0.804	0.620	Received
11	87380	65536	65536	60.02	8358.48	15.58	44.28	0.611	0.868	Transmit
12	87380	65536	65536	60.02	8352.43	20.54	31.58	0.806	0.619	Received
13	87380	65536	65536	60.01	8359.48	15.66	44.56	0.614	0.873	Transmit
14	87380	65536	65536	60.02	8352.01	20.51	31.54	0.805	0.619	Received

16.3 L2TPv3 Test Conclusions

Without a clear resolution to the performance issues with the Cisco ASR router, no conclusive results can be reported. However, it appears that without the ASR, host-to-host performance is not adversely impacted with the ABEx implementation of L2TPv3. Therefore, it is assumed that if a router is operating properly, the ABEx L2TPv3 protocol implementation should have no effect on traffic performance.

17 MSDPI INFINIBAND SERVER-CLIENT TEST

The following test was conducted to collect bandwidth performance data between a single IB server hardware platform, thus single IB interface, and multiple client hardware platforms. By using the ib_send_bw subsystem test utility, three MSDPI ib_send_bw test clients began sending IB QPs to the MSDPI ib_send_bw server at a predetermined time. For this test, startup was set to start at the same time from all the client platforms.

17.1 System Configuration Overview

Figure 43 illustrates the test configuration. This configuration consisted of two Dell R610s and two Dell R310s, named TD24, TD25, TD18, and TD30 for discussion purposes. One of the R610s, TD24, acted as the MSDPI ib_send_bw server and the other three systems (TD25, TD18, and TD30) functioned

as ib_send_bw clients. The R610s are Intel 5500 series processors with Mellanox PCIe Connect X MHZH29-XTR dual port IB/10GE network adapter cards. The specifications for this card are detailed in Table 17.

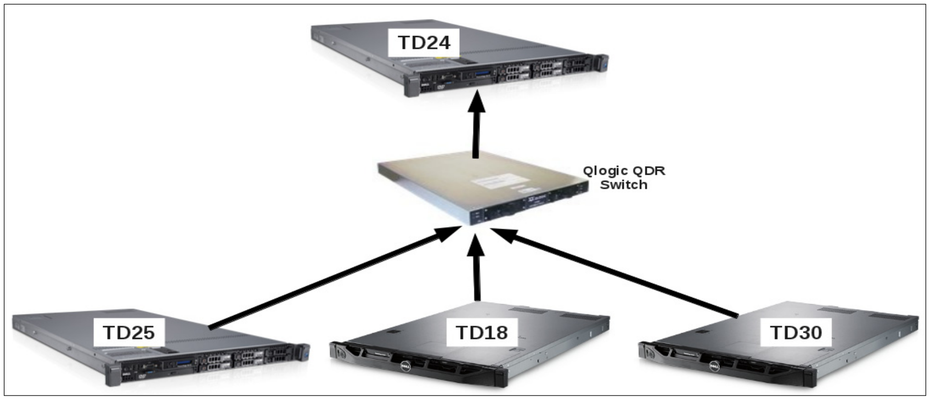


Fig. 43 — InfiniBand startup performance test

Table 17 — InfiniBand Network Adapter Specifications

InfiniBand:	IBTA v1.2.1, Auto-Negotiation (40Gb/s, 10Gb/s per lane), (20Gb/s, 5Gb/s per lane) or (10Gb/s, 2.5Gb/s per lane)
Ethernet:	IEEE Std 802.3ae 10 Gigabit Ethernet IEEE Std 802.3ad Link Aggregation and Failover IEEE Std 802.3x Pause IEEE Std 802.1Q VLAN tags IEEE Std 802.1p Priorities Multicast Jumbo frame support (10KB) 128 MAC/VLAN addresses per port
QoS:	8 Virtual Lanes for InfiniBand 8 Priority Queues for Ethernet
RDMA Support:	Yes, All Ports
Data Rate	
SFP+ Ethernet:	10 Gb/s
QSFP:	40 Gb/s
InfiniBand:	2.0 SERDES @ 5.0 GT/s
PCI Express:	

17.2 Test Script Overview

Table 18 contains the MSDPI server configuration commands issued to activate each of the MSDPI ib_send_bw test servers. The “icp” command prints diagnostic information on the MSDPI console which shows current packet processing information as outlined in Section 15. For this test, three concurrently executed MSDPI programs are run because the ib_send_bw subsystem, as of this writing, is not multi-thread safe. Therefore, each of the three clients will access a predetermined associated ib_send_bw server by using a separate, specifically assigned SIP and ib_send_bw port number.

Table 18 — Configuration Commands for Each MSDPI Server

```
$ sudo /usr/local/SIPCP/bin/msdpi --contact=sip:10.2.1.202:25060 --subsystemtag=APPL
--systemtag=APPL
MSDPI# icp sip:10.2.1.202:25060 1

$ sudo /usr/local/SIPCP/bin/msdpi --contact=sip:10.2.1.202:25061 --subsystemtag=APPL
--systemtag=APPL
MSDPI# icp sip:10.2.1.202:25061 1

$ sudo /usr/local/SIPCP/bin/msdpi --contact=sip:10.2.1.202:25061 --subsystemtag=APPL
--systemtag=APPL
MSDPI# icp sip:10.2.1.202:25061 1
```

The client configuration commands are shown in Table 19, Table 20, and Table 21. As with the server, the MSDPI command “icp” instructs the subsystem to display diagnostic information about packet processing. The command “idp” adds additional diagnostics by collecting, formatting, and then archiving the processing information to a log file. This log file can later be exported to a spreadsheet application for further analysis.

Table 19 — TD25 Configuration Command Entries

```
[TD25]# ./bin/msdpi --contact=sip:10.2.1.218:25060 --subsystemtag=APPL --systemtag=APPL
MSDPI# icp sip:10.2.1.218:25060 1
APPL> APPL> idp sip:10.2.1.218:25060 1 /usr/local/SIPCP/logs/13Jul11-0915
```

Table 20 — TD18 Configuration Command Entries

```
[TD18]# ./bin/msdpi --contact=sip:10.2.1.250:25061 --subsystemtag=APPL --systemtag=APPL
MSDPI# icp sip:10.2.1.250:25061 1
APPL> idp sip:10.2.1.250:25061 1 /usr/local/SIPCP/logs/13Jul11-0915
```

Table 21 — TD30 Configuration Command Entries

```
[TD30]# ./bin/msdpi --contact=sip:10.2.1.154:25062 --subsystemtag=APPL --systemtag=APPL
MSDPI# icp sip:10.2.1.154:25062 1
APPL> idp sip:10.2.1.154:25062 1 /usr/local/SIPCP/logs/13Jul11-915
```

Table 22 details the command directives issued by the MSDPI Test Master to each of the targeted test systems (both the ib_send_bw server and clients). Each of the files transmitted (as an MSDPI SIP SIMPLE message) contains the test parameter directives (in the MSDPI PIDF) to each of the ib_send_bw servers and ib_send_bw clients. Again, note, the ib_send_bw servers are multiple MSDPI instances on the same R610 hardware and each of these can run on separate hardware platforms.

Table 23 details the file contents for the servers and Table 24 for the clients. All the server and client directives files are basically the same with only the system address, port, and ib_send_bw port values changed. Each client's time tags are configured to the same start time.

Table 22 — MSDPI Test-Master Test Script Command Entries

```
MSDPI# saapplf sip:10.2.1.202:25060 APPL 0001 /usr/local/SIPCP/etc/TD24-19200-server-only-
ib_send_bw.xml
MSDPI# saapplf sip:10.2.1.202:25061 APPL 0001 /usr/local/SIPCP/etc/TD24-19201-server-only-
ib_send_bw.xml
MSDPI# saapplf sip:10.2.1.202:25062 APPL 0001 /usr/local/SIPCP/etc/TD24-19202-server-only-
ib_send_bw.xml
MSDPI# saapplf sip:10.2.1.218:25060 APPL 0001 /usr/local/SIPCP/etc/TD25-19200-client-
ib_send_bw.xml
MSDPI# saapplf sip:10.2.1.250:25061 APPL 0001 /usr/local/SIPCP/etc/TD18-19201-client-
ib_send_bw.xml
MSDPI# saapplf sip:10.2.1.154:25062 APPL 0001 /usr/local/SIPCP/etc/TD30-19202-client-
ib_send_bw.xml
```

Table 23 — MSDPI ib_send_bw Server Test Script Parameter Directives File


```
<?xml version="1.0" encoding="ISO-8859-1"?>
<system_commands>
<system>
<system_address>10.2.1.202</system_address>
<system_prefixlength>24</system_prefixlength>
<system_ttl>120</system_ttl>
<application>
<current_time>07-13-2011 08:00:00</current_time>
<current_time_usec>0000</current_time_usec>
<start_time>07-13-2011 08:00:00</start_time>
<start_time_usec>0000</start_time_usec>
<command>ib_send_bw</command>
<arguments>-p 19200</arguments>
<mode>server</mode>
</application>
<system_command_poc>
<name>NRL</name>
<phone>111-111-1111</phone>
</system_command_poc>
</system>
<checksum>SHA-123456789</checksum>
</system_commands>
```


Table 24 — MSDPI ib_send_bw Client Test Script Parameter Directives File


```
<?xml version="1.0" encoding="ISO-8859-1"?>
<system_commands>
<system>
<system_address>10.2.1.218</system_address>
<system_prefixlength>24</system_prefixlength>
<system_ttl>120</system_ttl>
<application>
<current_time>07-13-2011 09:15:00</current_time>
<current_time_usec>0000</current_time_usec>
<start_time>07-13-2011 09:15:00</start_time>
<start_time_usec>0000</start_time_usec>
<command>ib_send_bw</command>
<arguments>-p 19200 -h 10.2.1.202</arguments>
<mode>client</mode>
</application>
<system_command_poc>
<name>NRL</name>
<phone>111-111-1111</phone>
</system_command_poc>
</system>
<checksum>SHA-123456789</checksum>
</system_commands>
```

17.3 Raw Test Data

Table 25, Table 26, and Table 27 contain raw data samples from the ib_send_bw client test systems, TD25, TD18, and TD30.

Table 25 — Sampling of TD25 Raw Test Data

0714201106	0.609263	65536	1000	3207.08	3205.38
0714201106	0.090296	65536	1000	3203.86	3202.24
0714201106	0.454187	65536	1000	3204.85	3203.16
0714201106	0.868280	65536	1000	2913.13	1711.43
0714201106	0.397521	65536	1000	3205.03	3203.36
0714201106	0.764565	65536	1000	3204.97	3203.31
0714201106	0.215541	65536	1000	3205.03	3203.35
0714201106	0.652488	65536	1000	2995.10	1714.71
0714201106	0.095433	65536	1000	3204.97	3203.31
0714201106	0.637530	65536	1000	3206.58	3204.93
0714201106	0.160539	65536	1000	3205.03	3203.34
0714201106	0.663229	65536	1000	3204.17	3202.50
0714201106	0.101206	65536	1000	3205.22	3203.60
0714201106	0.546593	65536	1000	3205.03	3203.39
0714201106	0.990263	65536	1000	3204.85	3203.18
0714201106	0.455288	65536	1000	3205.47	3203.74
0714201106	0.741435	65536	1000	3205.16	3203.51
0714201106	0.106228	65536	1000	3205.03	3203.41
0714201106	0.551517	65536	1000	3202.87	3201.27
0714201106	0.045490	65536	1000	3205.10	3203.47
0714201106	0.477092	65536	1000	3205.28	3203.65
0714201106	0.967527	65536	1000	3204.91	3203.22
0714201106	0.414128	65536	1000	3205.22	3203.60
0714201106	0.965537	65536	1000	3205.28	3203.60
0714201106	0.375228	65536	1000	3204.66	3202.97
0714201106	0.791527	65536	1000	3203.98	3202.37
0714201106	0.319448	65536	1000	3205.03	3203.37
0714201106	0.686031	65536	1000	3205.53	3203.85
0714201106	0.135537	65536	1000	3205.16	3203.50
0714201106	0.686875	65536	1000	3202.63	2165.21
0714201106	0.063536	65536	1000	3205.10	3203.44
0714201106	0.453276	65536	1000	2728.41	1706.81
0714201106	0.911633	65536	1000	3204.85	3203.21
0714201106	0.283328	65536	1000	3204.73	3203.08
0714201106	0.754639	65536	1000	3204.91	3203.27
0714201106	0.149205	65536	1000	3205.10	3203.47
0714201106	0.697527	65536	1000	3203.92	3202.32
0714201106	0.102147	65536	1000	3207.26	3205.44
0714201106	0.532335	65536	1000	3205.47	3203.78

Table 26 — Sampling of TD18 Raw Test Data

0714201106	0.530554	65536	1000	1887.00	1886.98
0714201106	0.988534	65536	1000	1730.72	1730.72
0714201106	0.383229	65536	1000	3206.24	3204.61
0714201106	0.869228	65536	1000	3206.24	3204.58
0714201106	0.276967	65536	1000	3206.38	3204.75
0714201106	0.813975	65536	1000	3206.24	3204.56
0714201106	0.261216	65536	1000	3206.17	3204.51
0714201106	0.710624	65536	1000	1206.19	1206.19
0714201106	0.171515	65536	1000	3206.31	3204.67
0714201106	0.629222	65536	1000	3206.45	3204.76
0714201106	0.042224	65536	1000	3206.17	3204.55
0714201106	0.527228	65536	1000	3206.11	3204.45
0714201106	0.898241	65536	1000	3205.97	3204.28
0714201106	0.361816	65536	1000	3206.17	3204.49
0714201106	0.820216	65536	1000	3206.52	3204.87
0714201106	0.350225	65536	1000	3206.17	3204.47
0714201106	0.877161	65536	1000	3205.97	3204.34
0714201106	0.445205	65536	1000	3206.11	3204.45
0714201106	0.938605	65536	1000	1701.57	1701.57
0714201106	0.305242	65536	1000	3206.24	3204.61
0714201106	0.726167	65536	1000	3204.73	3197.98
0714201106	0.093256	65536	1000	3206.24	3204.57
0714201106	0.461260	65536	1000	3206.11	3204.36
0714201106	0.843026	65536	1000	2138.78	2138.75
0714201106	0.189748	65536	1000	3204.80	3203.16
0714201106	0.669289	65536	1000	3205.76	3204.07
0714201106	0.115138	65536	1000	3206.17	3204.48
0714201106	0.562282	65536	1000	3206.11	3204.43
0714201106	0.972065	65536	1000	3205.28	3197.67
0714201106	0.375242	65536	1000	3206.24	3204.53
0714201106	0.853318	65536	1000	3206.11	3204.46
0714201106	0.368227	65536	1000	3206.04	3204.36
0714201106	0.737896	65536	1000	3198.84	3197.33
0714201106	0.119207	65536	1000	3206.04	3204.35
0714201106	0.532148	65536	1000	3206.24	3204.61
0714201106	0.912224	65536	1000	3206.17	3204.49
0714201106	0.481330	65536	1000	3203.09	3197.62
0714201106	0.891216	65536	1000	3206.11	3204.40
0714201106	0.337228	65536	1000	3206.11	3204.39

Table 27 — Sampling of TD30 Raw Test Data

0714201106	0.524646	65536	1000	3193.89	1884.77
0714201106	0.991458	65536	1000	3110.61	1728.73
0714201106	0.426906	65536	1000	3199.08	3194.02
0714201106	0.801497	65536	1000	3200.31	3198.51
0714201106	0.239535	65536	1000	3200.45	3198.61
0714201106	0.690697	65536	1000	3200.58	3198.79
0714201106	0.143489	65536	1000	3200.51	3198.70
0714201106	0.707965	65536	1000	3185.66	1274.11
0714201106	0.210500	65536	1000	3200.65	3198.86
0714201106	0.591458	65536	1000	3200.65	3198.84
0714201106	0.092456	65536	1000	3200.45	3198.63
0714201106	0.626259	65536	1000	3187.63	1838.63
0714201106	0.102481	65536	1000	3200.58	3198.77
0714201106	0.656687	65536	1000	3199.49	3190.17
0714201106	0.106635	65536	1000	3199.69	3197.90
0714201106	0.495810	65536	1000	3200.58	3198.77
0714201106	0.947475	65536	1000	3200.65	3198.82
0714201106	0.367142	65536	1000	3200.58	3198.77
0714201106	0.942057	65536	1000	1701.12	1701.11
0714201106	0.502477	65536	1000	3200.38	3198.56
0714201106	0.888724	65536	1000	3199.01	2201.42
0714201106	0.341469	65536	1000	3199.76	3197.98
0714201106	0.838187	65536	1000	3199.01	2135.32
0714201106	0.258691	65536	1000	3200.65	3198.84
0714201106	0.713832	65536	1000	3199.83	3198.09
0714201106	0.253916	65536	1000	3194.91	1910.23
0714201106	0.730165	65536	1000	3200.58	3198.75
0714201106	0.214758	65536	1000	3200.51	3198.70
0714201106	0.707476	65536	1000	3200.45	3198.60
0714201106	0.290457	65536	1000	3200.45	3198.69
0714201106	0.822458	65536	1000	3200.24	3198.42
0714201106	0.423476	65536	1000	3200.58	3198.84
0714201106	0.956022	65536	1000	3200.17	3198.33
0714201106	0.363948	65536	1000	3200.24	3198.41
0714201106	0.854810	65536	1000	3200.45	3198.64
0714201106	0.310500	65536	1000	3200.38	3198.54
0714201106	0.722478	65536	1000	3200.58	3198.74
0714201106	0.179676	65536	1000	3200.51	3198.68
0714201106	0.624819	65536	1000	3200.38	3198.55

17.4 Test Data Analysis

While this test was only for a short duration, the raw data confirms all systems began the ib_send_bw test within a second of each other but not at the precise microsecond. Further, it appears the first system, TD25, reached full bandwidth use within its first iteration cycle while TD18 and TD30 began at a lower bandwidth allocation but quickly settled into full use within the first or second iteration cycle. Table 28 contains the first three iteration cycles from TD25, TD18, and TD30. Two runs for each system are shown. Figure 44 shows the performance at startup and Fig. 45, Fig. 46, and Fig. 47 reflect the first 39 iterations from each of the clients. In the data charts, the horizontal axis is time and the vertical axis is bandwidth.

Table 28 — First Three Iteration Cycle Data

```
TD25 Run 1
0713201109 0.365101  65536 1000 3200.16 3198.55
0713201109 0.755907  65536 1000 3204.91 3203.20
0713201109 0.132535  65536 1000 3205.34 3203.68
TD25 Run 2
0714201106 0.609263  65536 1000 3207.08 3205.38
0714201106 0.090296  65536 1000 3203.86 3202.24
0714201106 0.454187  65536 1000 3204.85 3203.16

TD18 Run 1
0713201109 0.423747  65536 1000 2887.18 1720.69
0713201109 0.890068  65536 1000 3206.17 3204.50
0713201109 0.294175  65536 1000 3206.04 3204.35
TD18 Run 2
0714201106 0.530554  65536 1000 1887.00 1886.98
0714201106 0.988534  65536 1000 1730.72 1730.72
0714201106 0.383229  65536 1000 3206.24 3204.61

TD30 Run 1
0713201109 0.423264  65536 1000 1723.58 1723.56
0713201109 0.801590  65536 1000 3200.38 3198.54
0713201109 0.176824  65536 1000 3199.35 3197.55
TD30 Run 2
0714201106 0.524646  65536 1000 3193.89 1884.77
0714201106 0.991458  65536 1000 3110.61 1728.73
0714201106 0.426906  65536 1000 3199.08 3194.02
```

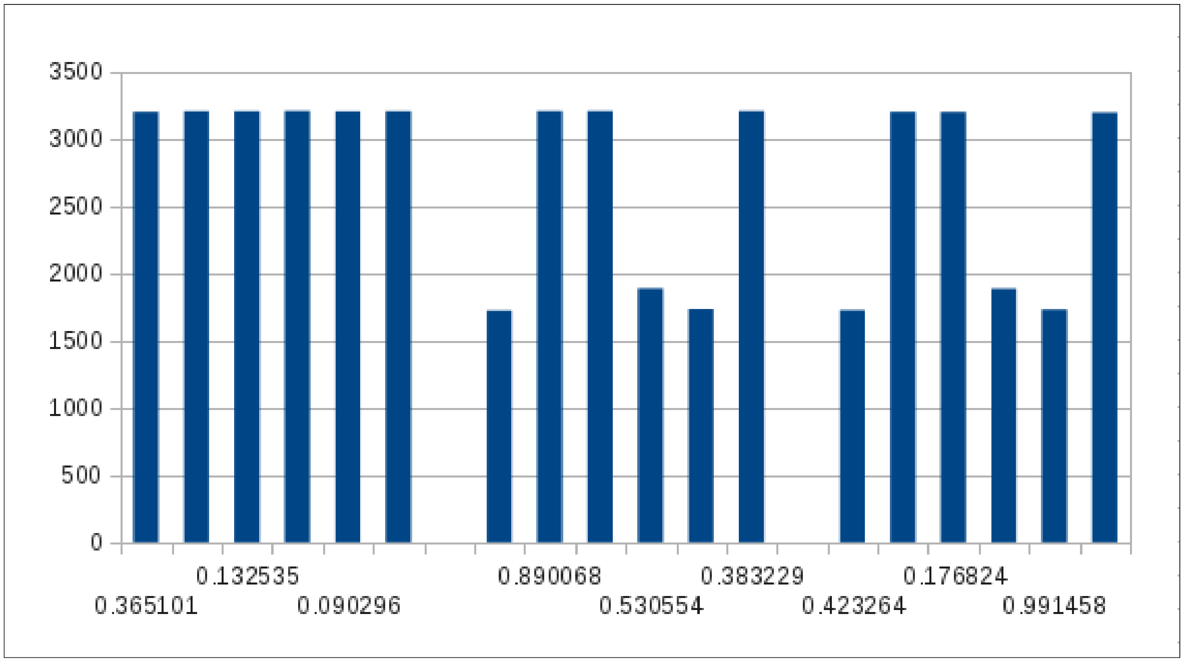


Fig. 44 — Startup comparison data

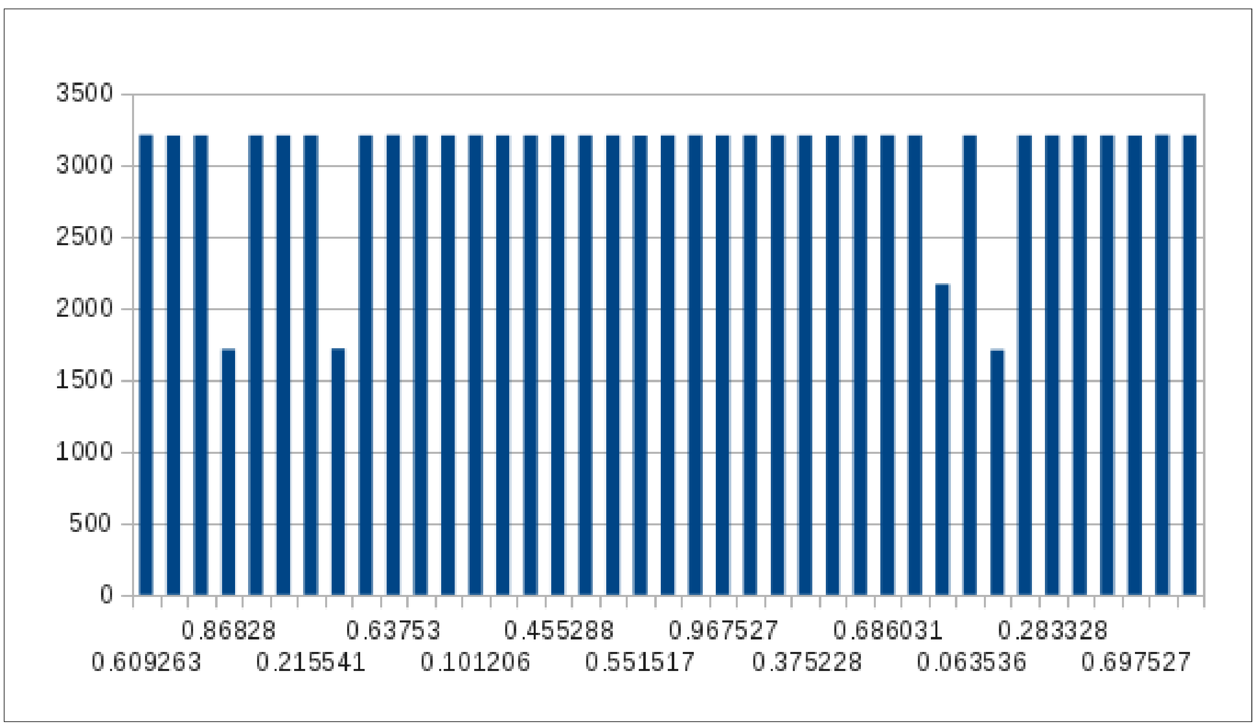


Fig. 45 — TD25 test data

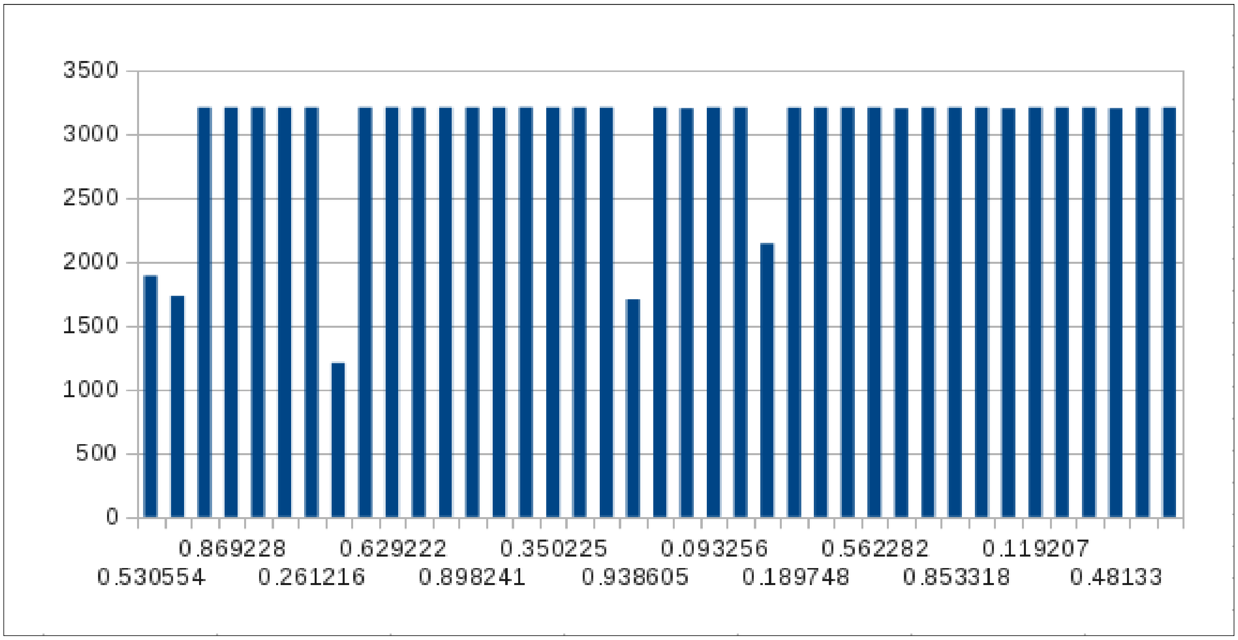


Fig. 46 — TD18 test data

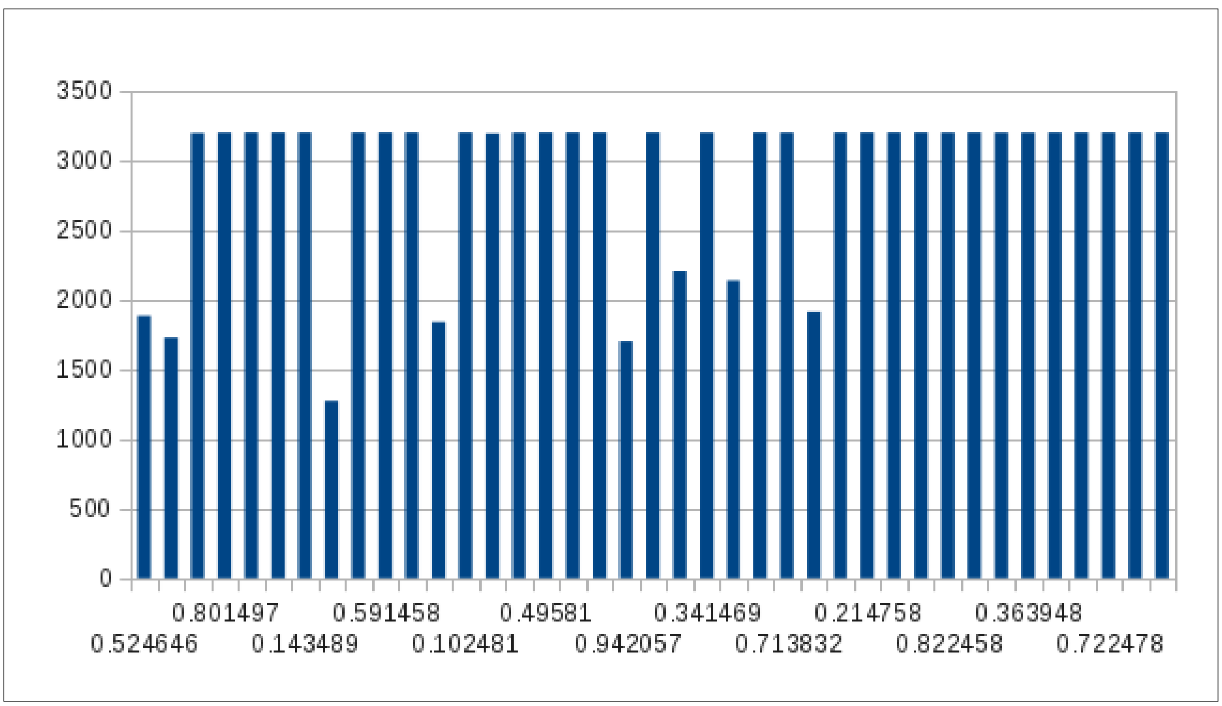


Fig. 47 — TD30 test data

17.5 Rerunning the Test Data Analysis

The previous test demonstrated some problems with the Dell R310 servers, so the tests were executed again after some hardware changes. The Dell R310s were replaced with Dell R610s. Table 29 and Fig. 48, Fig. 49, Fig. 50, and Fig. 51 report the test data results (horizontal axis is time, vertical is bandwidth).

Table 29 — First Three Iteration Cycles Reevaluated

<u>TD25 Reevaluation Run 1</u>
393376 3203.56
839052 3203.55
322047 3203.33
<u>TD26 Reevaluation Run 1</u>
06376426 3198.97
06811610 3204.34
06294440 3204.70
<u>TD28 Reevaluation Run 1</u>
481924 3193.51
957261 3204.43
440251 3204.66

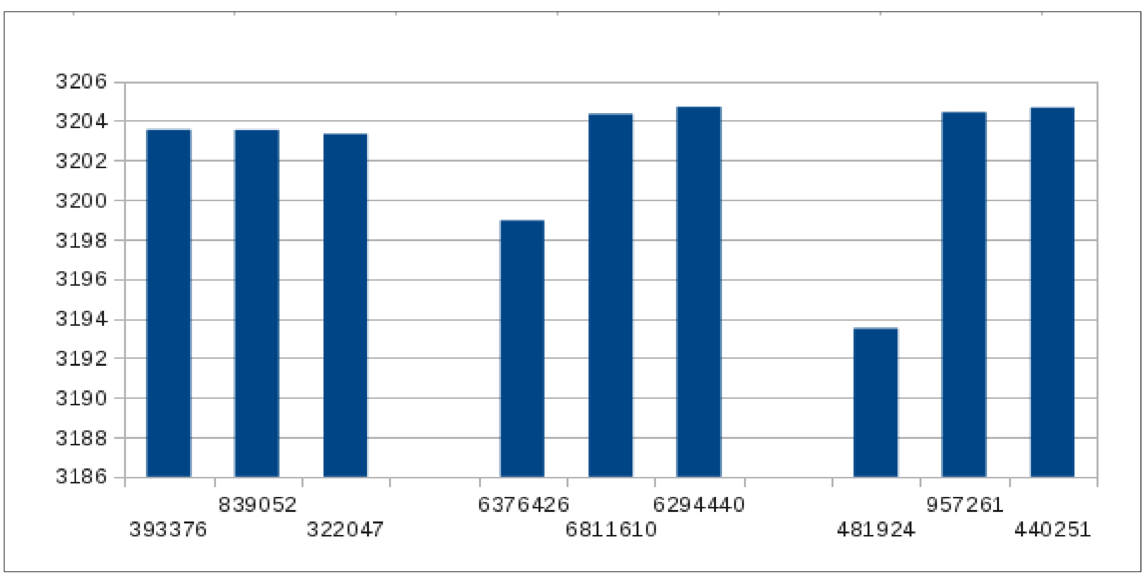


Fig. 48 — Reevaluation of the startup comparison data

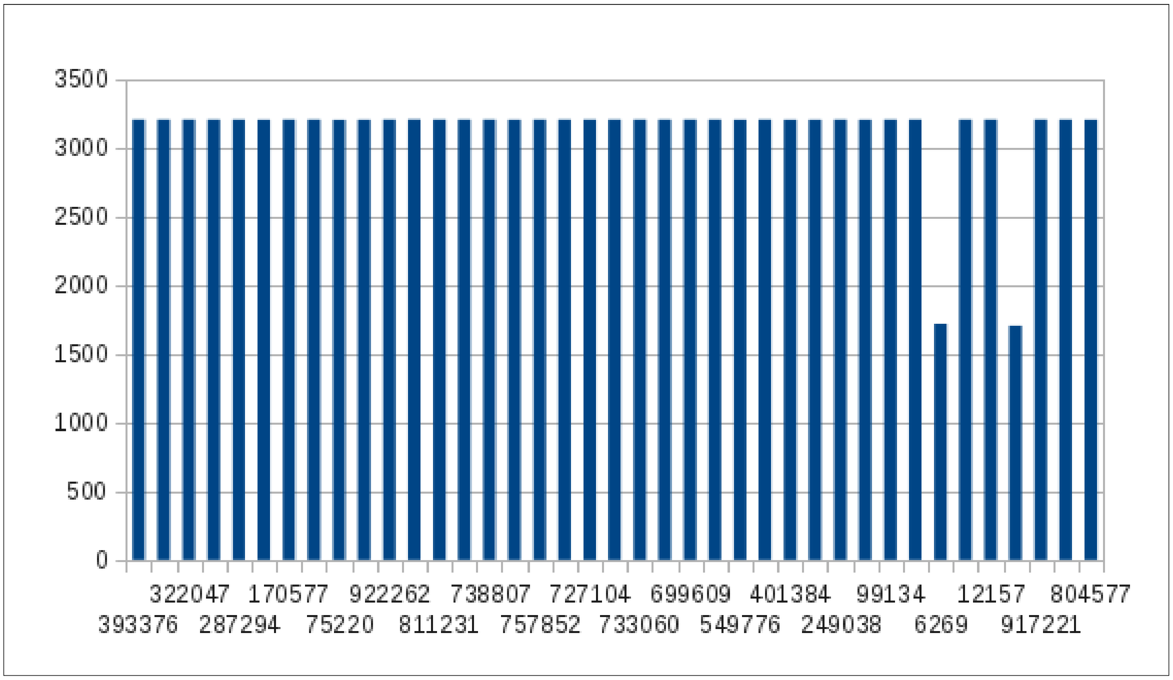


Fig. 49 — TD25 re-test data

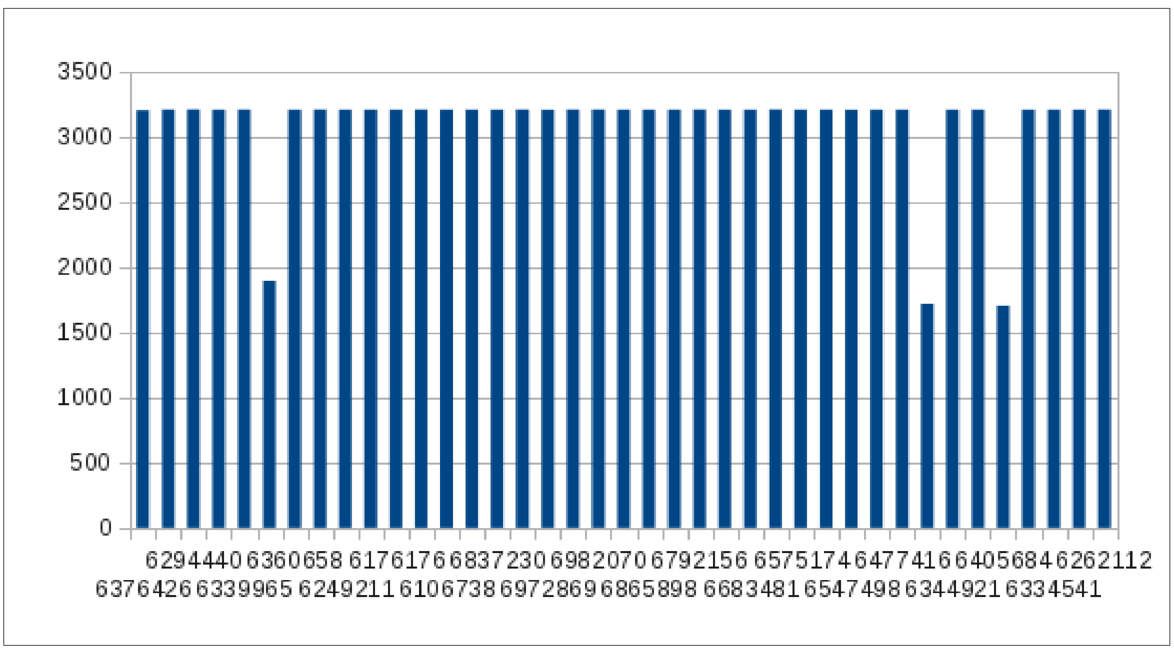


Fig. 50 — TD26 re-test data

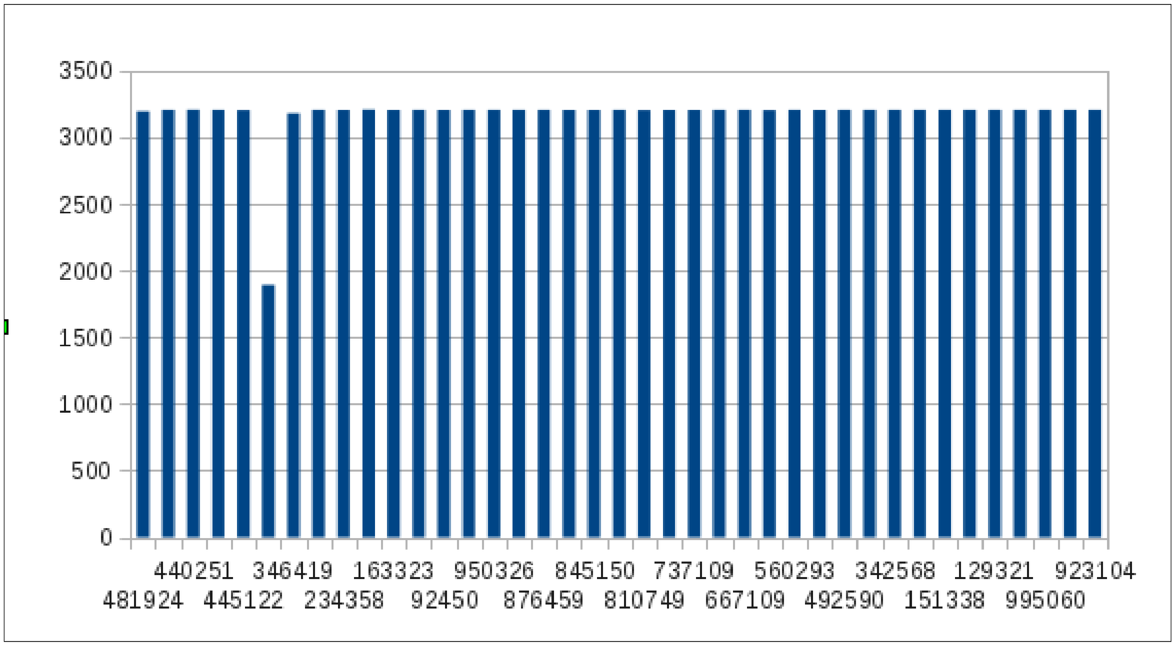


Fig. 51 — TD28 re-test data

17.6 Test Data Conclusion

It appears that the R610, with its faster multi-threaded processor, has advantages for initial IB packet processing. But caution must be observed when analyzing this data, as pristine laboratory controls over system heat, hardware characteristics, power fluctuations, and external traffic such as SNMP and SSH sessions, are only a few factors that may contribute to performance. For example, the MSDPI clients were remotely accessed, which meant an SSH session was in progress. Also, IB uses IP/TCP to set up a QP transfer. This further delays IB when TCP connections are delayed due to system resource allocation issues. Although using a separate interface, system resources (CPU, memory, and PCIe bus) were still being used. Further, there was no way to accurately measure the impact of the InfiniBand switch on the test performance. For these tests, it was observed that approximately 10% of IB interface performance is impacted by these "other" resources. To obtain test data not affected by outside influences, systems outside the funding scope of this program would be required. For normal DoD/IC operational configurations for systems that do not have critical timing issues, these test results are more than adequate.

18 MSDPI ENGINEERED INTO A LIVECD DISTRIBUTION

18.1 Linux LiveCD

A LiveCD is a Linux custom-configured bootable CD or DVD which when booted runs a complete operating system without requiring a secondary storage device such as a hard drive. In this case, a Fedora 15 (or 14) bootable DVD image (ISO format) contains the MSDPI system, the directory tree /usr/local/SIPCP, and several MSDPI XML configuration files.

18.2 MSDPI LiveCD

The reason for creating an MSDPI LiveCD is to have a test environment that has control of the operating environment, taking full advantage of the rich feature set of a Fedora 15/14 MSDPI environment yet still preserving the existing system's previous operating system state. For example, a typical DoD Linux server runs a Red Hat 5.5 distribution specifically configured for a particular agency mission. Sometimes when debugging I/O hardware performance problems, the interface in question may require test configurations that are best implemented without interference from other synchronous running programs. This dedicates system resources, such as I/O and processor cycles, to the test program. Test scenarios can be executed without delays from other applications. Further, a LiveCD establishes the test tool has a highly mobile diagnostic utility, reducing deployment time. For example, the need to build multiple versions of MSDPI that are compatible with every version of Linux distribution is not required. Further, not every Linux distribution includes all the open source runtime libraries that MSDPI exploits. For example, Sofia-SIP is MSDPI's core SIP processing system. While Fedora supports a native installation, other distributions do not include it, therefore requiring Sofia-SIP to be compiled for each of those distributions. Another advantage is the ability to keep the MSDPI system up to date with the latest releases not only of MSDPI changes but of the exploited open source libraries. Further, enhancements of newly developed technologies, such as the Bay Microsystems L2TPv3 test discussed in this report, are quickly integrated into the MSDPI system. Since MSDPI has been set up as a LiveCD, added security can easily be incorporated by locking down the distribution so that only MSDPI is accessible and further enhanced to allow only a particular test configuration to be performed. In so doing, not only is security improved for testing, but any user with no test experience can be instructed to boot the LiveCD, while the experienced tester tests and subsequently retrieves the resulting test data remotely, as discussed in Section 15.

18.3 MSDPI LiveCD Feature Sets

MSDPI's LiveCD current feature set includes the following:

1. "perftest" "send_bw" test subsystem which performs bandwidth diagnostics between two or more IB hosts.
2. Any shell run-able Linux command, for example, Netperf.

18.4 How to Build an MSDPI LiveCD

The following steps detail how an MSDPI bootable LiveCD is created, assuming the development environment is Fedora 14 (or 15) and the proper rpmbuild environment has been initialized.

1. Create an rpm repository for the MSDPI system. This is required because the LiveCD process retrieves the MSDPI system from the repository.

a. Create a yum repository file named “msdpi.repos” and place it in the directory /etc/yum.repos.d/. Figure 52 illustrates the contents of the file.
b. Create a repository directory and run the command: createrepo /path-to-your-msdpi-repository. Typically, this path is within the http server (in this case Cherokee) directory tree, for example: /var/www/cherokee/msdpi-repos.

2. Now create an MSDPI rpm file to be used by the LiveCD build process.

a. Create rpmbuild specification files with the information detailed in Fig. 53 and Fig. 54.
b. Place the specification files in the appropriate rpmbuild directory, ~/rpmbuild/SPECS.
c. Place a copy of the MSDPI archive files in the appropriate rpmbuild directory, ~/rpmbuild/SOURCES.

3. The name of these files and the directories they archived must match the filename designation specified in the rpmbuild specification files.

a. Run the command line sequence: rpmbuild -bb –target=`uname -m` ./[msdpi specification file].
b. Copy the built rpm files to the MSDPI repository created in the above yum repository creation instructions (step 1).
c. Recreate the repomd metadata file by reissuing step 1b from the yum creation process above.

4. The last thing to do is build the LiveCD ISO image and burn onto a DVD or USB drive. Since the size of the ISO file is more than 640 MB, a DVD must be used.

a. Create a kickstart file with the data shown in Fig. 55.
b. Edit the kickstart file to reflect the interfaces and hostname of the target system.
c. Issue the command to create the ISO image: livecd-creator --config=msdpi.ks --fslabel=MSDPI –cache=/var/cache/live.
d. Burn the ISO image MSDPI.iso to DVD using any installed DVD writer program (for example, GnomeBaker). Typically, by placing a blank DVD in the DVD writer, the default DVD writer program auto-loads. If a USB bootable LiveCD is being created, issue the command line sequence: livecd-iso-to-disk --format --reset-mbr MSDPI.iso /dev/[USB device].

```
[msdpi]
name=local
baseurl=http://localhost/msdpi-repos/
enabled=1
gpgcheck=0
```

Fig. 52 — Example of the MSDPI repository file

```
Name:       msdpi
Version:    30.06.11
Release:    1%{?dist}
Summary:    Multi-Service Domain Protecting Interface
License:    GPL+
URL:        http://localhost/develop/SIPCONTROLPLANE
Source0:    http://localhost/develop/SIPCONTROLPLANE/%{name}-%{version}.tar.gz
# BuildRequires:
# Requires:
%description
The MSDPI program, SIP Control Plane.
%prep
%setup -q
%build
%configure --prefix=/usr/local/SIPCP
make %{?_smp_mflags}
%install
rm -rf $RPM_BUILD_ROOT
# make install DESTDIR=$RPM_BUILD_ROOT
mkdir -p $RPM_BUILD_ROOT/usr/local/SIPCP/bin
cp %{_builddir}/%{name}-%{version}/src/msdpi $RPM_BUILD_ROOT/usr/local/SIPCP/bin
%clean
rm -rf $RPM_BUILD_ROOT
%files
%defattr(-,root,root,-)
%doc
/usr/local/SIPCP/bin/msdpi
%changelog
```

Fig. 53 — Example of the MSDPI rpmbuild specification file

```
Name:          msdpi-packages
Version:       30.06.11
Release:       1%{?dist}
Summary:    Multi-Service Domain Protecting Interface Packages
License:      GPL+
URL:            http://localhost/develop/SIPCONTROLPLANE
Source0:        http://localhost/develop/SIPCONTROLPLANE/%{name}-%{version}.tar.gz
# BuildRequires:
# Requires:
%description
The "MSDPI program, SIP Control Plane packages includes those necessary files to build a complete LiveCD system
For example, adding various configuration files:  ifcfg-eth?, ifcfg-ib?, openvpn config and key files.
NOTE: Each of the LiveCDs are built to represent a specific system configuration to include the system name,
interface parametrers (aka IP address), openvpn files, specific files in the etc/sysconfig directory.
%prep
%build
%install
rm -rf $RPM_BUILD_ROOT
mkdir -p $RPM_BUILD_ROOT/usr/sbin
mkdir -p $RPM_BUILD_ROOT/etc/rdma
mkdir -p $RPM_BUILD_ROOT/etc/init.d
mkdir -p $RPM_BUILD_ROOT/usr/local/sbin
mkdir -p $RPM_BUILD_ROOT/usr/local/bin
mkdir -p $RPM_BUILD_ROOT/usr/local/SIPCP
cp -R /develop/LIVECD/SIPCP/* $RPM_BUILD_ROOT/usr/local/SIPCP
cp -R /develop/LIVECD/ETC/rdma/* $RPM_BUILD_ROOT/etc/rdma
cp /develop/LIVECD/ETC/init.d/rdma $RPM_BUILD_ROOT/etc/init.d/rdma
cp /usr/sbin/opensm $RPM_BUILD_ROOT/usr/sbin/opensm
%clean
rm -rf $RPM_BUILD_ROOT
%files
%defattr(-,root,root,-)
%doc
/usr/local/SIPCP/bin/ib_clock_test
/usr/local/SIPCP/bin/ib_rdma_bw
/usr/local/SIPCP/bin/ib_rdma_lat
/usr/local/SIPCP/bin/ib_read_bw
/usr/local/SIPCP/bin/ib_read_lat
/usr/local/SIPCP/bin/ib_send_bw
/usr/local/SIPCP/bin/ib_send_lat
/usr/local/SIPCP/bin/ib_write_bw
/usr/local/SIPCP/bin/ib_write_bw_postlist
/usr/local/SIPCP/bin/ib_write_lat
/etc/init.d/rdma
/etc/rdma/fixup-mtrr.awk
/etc/rdma/mlx4.conf
/etc/rdma/opensm.conf
/etc/rdma/rdma.conf
/etc/rdma/setup-mlx4.awk
/usr/sbin/opensm
/usr/local/SIPCP/sbin/netperf.sh
/usr/local/SIPCP/bin/netperf
/usr/local/SIPCP/bin/netserver
/usr/local/SIPCP/bin/msdpi
/usr/local/SIPCP/sbin/msdpiclientsh
/usr/local/SIPCP/sbin/msdpish
/usr/local/SIPCP/etc/SITE-A_server-only-ib_send_bw.xml
%changelog
```

Fig. 54 — Example of the MSDPI package rpmbuild specification file

```
%include /usr/share/spin-kickstarts/fedora-livecd-desktop.ks
repo --name=local --baseurl=http://localhost/msdpi-repos/
%packages
msdpi-packages
%end
%post
# FIXME: it’d be better to get this installed from a package
cat > /etc/sysconfig/network-scripts/ifcfg-eth0 << EOF
DEVICE=eth0
#HWADDR=00:22:68:1E:E2:80
ONBOOT=yes
NM_CONTROLLED=yes
IPADDR=10.128.112.6
BOOTPROTO=none
NETMASK=255.255.255.0
TYPE=Ethernet
GATEWAY=10.128.112.1
IPV6INIT=yes
USERCTL=yes
PREFIX=24
DNS1=10.1.1.1
EOF
cat > /etc/sysconfig/network << EOF
NETWORKING=yes
HOSTNAME=FARP.atd.net
NTPSERVERARGS=iburst
NOZEROCONF=yes
EOF
cat > /etc/sysconfig/vncservers << EOF
VNCSERVERS=“50:msdpi-user”
EOF
%end
```

Fig. 55 — Example of the MSDPI package LiveCD kickstart file

19 WORK OUTSTANDING

There are still many enhancements and improvements to be made to MSDPI, including the following.

1. In the original FY11 project plan established by the sponsor, required network components were to be purchased by the sponsor and delivered to NRL so that certain tests could be completed. To date, the equipment has not been delivered, preventing NRL from conducting planned tests such as the following:
    a. IB-to-WAN scenario testing as it relates to DoD/IC network configurations.
    b. Synchronized IB timing tests between multiple end systems.
    c. File system over 40G IB interface network tests using the DoD-compliant OS, Red Hat 5.5. NRL subscription licenses for Red Hat Server/Client software have expired, further impacting test execution.

2. Complete several IB test tool modifications to take full advantage of NRL's enhanced perftest modifications. These IB test tool modifications were not scoped within the FY11 tasking but have since been determined to be of importance for inclusion in any further follow-on work.
   a. Apply "send_bw" program fixes to the other IB tool sets (read_bw, read_lat, write_bw, write_lat, rdma_bw, rdma_lat, send_lat).
   b. Modify the IB tool sets for threaded operation.
   c. Complete the integration of the rest of the IB tool sets within MSDPI. Currently only "send_bw" is fully integrated.
   d. Include other IP test tools so they are fully integrated into MSDPI (not as subsystem shell commands). For example and most important is the full integration of the Netperf tool set. Other examples include incorporating tools such as ping, modifying SIP SIMPLE MESSAGE messages as ping-like exchanges, and incorporating Test TCP (TTCP), which is a benchmarking tool to measure TCP network performance, and other tools such as traceroute.
   e. Rework the command line so shell commands can be run dynamically (currently shell commands are compile time inclusions to MSDPI).

3. Complete SNMP processing to support basic MIB database exchanges and unique security features such as inter-domain exchanges. This will provide a solution for network product vendors, such as Anagran, who could exploit the MSDPI "Sandwich" within DoD/IC protected network configurations, thus seamlessly supporting product MIB inter-domain exchange.

4. Complete the incorporation of MSDPI within the Bay Microsystems product line by providing assistance to Bay with the Buildroot process.

5. Integrate the OpenFlow forwarding table process into MSDPI. This would provide the DoD/IC with a valuable full-featured and dynamically flexible network virtualization and simulation tool as well as a like encryption device development tool which does not exist today. It would also allow network encryption device and network component developers and test engineers to test and evaluate DoD/IC unique requirements without disrupting operational networks. Finally, it would provide a prototype reference for developers in meeting DoD/IC mission requirements.

20 CONCLUSION

The successful implementation of MSDPI prototypes provides solid proof of the many benefits this system has for the government. It provides an effective way to maintain the technological readiness of encryption device technology. It enhances the capabilities of existing information assurance systems. It provides a deployable mechanism for policy control between protected domains. It clearly demonstrates the ability to converge routing protocols, IA, and test sets into one control plane. Finally, the prototypes clearly demonstrate reduced costs and logistical management of DoD/IC operational encryption devices, test tool devices, and policy management devices.

REFERENCES

1. C.L. Robson, “Multi-Service Domain Protecting Interface Architecture,” NRL/FR/5591—08-10,176, Naval Research Laboratory, Washington, DC, December 19, 2008.
2. C.L. Robson, “Session Initiation Protocol Network Encryption Device Plain Text Domain Discovery Service,” NRL/FR/5591—07-10,156, Naval Research Laboratory, Washington, DC, December 7, 2007.
3. C.L. Robson, “How to Use FASTLANEs to Protect IP Networks,” NRL/MR/5590—06-8979, Naval Research Laboratory, Washington, DC, August 18, 2006.
4. RFC 3438, “Layer Two Tunneling Protocol (L2TP) Internet Assigned Numbers Authority (IANA) Considerations Update,” W. Townsley, 5 pp., The Internet Society, December 2002, http://tools.ietf.org/rfc/rfc3438.txt.
5. RFC 3986, “Uniform Resource Identifier (URI): Generic Syntax,” T. Berners-Lee, R. Fielding, and L. Masinter, 61 pp., The Internet Society, January 2005, www.ietf.org/rfc/rfc3986.txt.
6. RFC 1349, “Type of Service in the Internet Protocol Suite,” P. Almquist, 28 pp., The Internet Society, July 1992, http://tools.ietf.org/rfc/rfc1349.txt.
7. RFC 2547bis, “BGP/MPLS IP VPNs,” E.C. Rosen and Y. Rekhter, 49 pp., The Internet Society, October 2004, http://tools.ietf.org/id/draft-ietf-13vpn-rfc2547bis-03.txt.
8. RFC 2401, “Security Architecture for the Internet Protocol,” S. Kent and R. Atkinson, 66 pp., The Internet Society, November 1998, www.ietf.org/rfc/rfc2401.txt.

BIBLIOGRAPHY

Almesberger, W., “Linux Traffic Control – Next Generation,” October 18, 2002, available at http://tcng.sourceforge.net/doc/tcng-overview.pdf.

“Extensible Markup Language (XML) 1.0 (Fourth Edition),” T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, and F. Yergeau, September 2006, http://www.w3.org/TR/2006/REC-xml-20060816/.

IceWalkers, IOCTL Man Page, pp. 1, April 9, 2008.

Kashyap, V., and H.K.J. Chu, “IP encapsulation and address resolution over InfiniBand networks,” 13 pp. The Internet Society, February 6, 2002, http://tools.ietf.org/id/draft-ietf-ipoib-ip-over-infiniband-00.txt.

Mellanox Technologies Inc., InfiniBand Architecture Overview, pp. 1-44, April 9, 2008.

RFC 2327, “SDP: Session Description Protocol,” M. Handley and V. Jacobson, 42 pp., The Internet Society, April 1998, http://www.ietf.org/rfc/rfc2327.txt.

RFC 3087, “Control of Service Context using SIP Request-URI,” B. Campbell and R. Sparks, 39 pp., The Internet Society, April 2001, http://www.rfc-editor.org/rfc/rfc3087.txt.

RFC 3261, “SIP: Session Initiation Protocol,” J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, 269 pp., The Internet Society, June 2002, http://www.ietf.org/rfc/rfc3261.txt.

RFC 3264, “An Offer/Answer Model with the Session Description Protocol (SDP),” J. Rosenberg and H. Schulzrinne, 25 pp., The Internet Society, June 2002, http://www.ietf.org/rfc/rfc3264.txt.

RFC 3265, "Session Initiation Protocol (SIP)-Specific Event Notification," A.B. Roach, 38 pp., The Internet Society, June 2002, http://www.ietf.org/rfc/rfc3265.txt.

RFC 3270, "Multi-Protocol Label Switching (MPLS) Support of Differentiated Services," F. Le Faucheur (ed.), L. Wu, B. Davie, S. Davari, P. Vaananen, R. Krishnan, P. Cheval, and J. Heinanen, 64 pp., The Internet Society, May 2002, http://www.ietf.org/rfc/rfc3270.txt.

RFC 3327, "Session Initiation Protocol (SIP) Extension Header Field for Registering Non-Adjacent Contacts," D. Willis and B. Hoeneisen, 17 pp., The Internet Society, December 2002, http://www.rfc-editor.org/rfc/rfc3327.txt.

RFC 3515, "The Session Initiation Protocol (SIP) Refer Method," R. Sparks, 23 pp., The Internet Society, April 2003, http://www.ietf.org/rfc/rfc3515.txt.

RFC 3564, "Requirements for Support of Differentiated Services-aware MPLS Traffic Engineering," F. Le Faucheur and W. Lai, 22 pp., The Internet Society, July 2003, http://www.ietf.org/rfc/rfc3564.txt.

RFC 3856, "A Presence Event Package for the Session Initiation Protocol (SIP)," J. Rosenberg, 27 pp., The Internet Society, August 2004, http://www.ietf.org/rfc/rfc3856.txt.

RFC 3859, "Common Profile for Presence (CPP)," J. Peterson, 15 pp., The Internet Society, August 2004, http://tools.ietf.org/rfc/rfc3859.txt.

RFC 3863, "Presence Information Data Format (PIDF)," H. Sugano, S. Fujimoto, G. Klyne, A. Bateman, W. Carr, and J. Peterson, 28 pp., The Internet Society, August 2004, http://www.ietf.org/rfc/rfc3863.txt.

RFC 4105, "Requirements for Inter-Area MPLS Traffic Engineering," J.-L. Le Roux, J.-P. Vasseur, and J. Boyle, 22 pp., The Internet Society, June 2005, http://www.ietf.org/rfc/rfc4105.txt.

RFC 4303, "IP Encapsulating Security Payload (ESP)," S. Kent, 44 pp., The Internet Society, December 2005, http://www.ietf.org/rfc/rfc4303.txt.

RFC 4762, "Virtual Private LAN Service (VPLS) Using Label Distribution Protocol (LDP) Signaling," M. Lasserre and V. Kompella, 31 pp., The Internet Society, January 2007, http://tools.ietf.org/rfc/rfc4762.txt.

Sofia-SIP Library, http://sofia-sip.sourceforge.net.